INTRODUCTION TO T-BUG

by Don Inman & Kurt Inman

INTRODUCTION TO T-BUG

INTRODUCTION TO

T-BUG

THE TRS-80 MACHINE
LANGUAGE MONITOR

By Don and Kurt Inman

dilithium Press
Portland, Oregon

10 9 8 7 6 5 4 3 2 1

ISBN: 0-918398-33-9
Library of Congress catalog card number: 79-67461

Printed in the United States of America

dilithium Press
P.O. Box 92
Forest Grove, OR 97116

This book is dedicated to Bob Albrecht, who opened up the wonderful world of computers to both of us. Although he professes a violent dislike for machine language programming, the Dragon strongly supports the idea of "de-mystifying" computers. This "de-mystifying" has been our aim in writing this book.

The Authors

TABLE OF CONTENTS

INTRODUCTION

The computer is a tool for solving problems that come in various forms—educational, recreational, vocational, etc. Learning to program a computer is a meaningless task unless it is approached in a problem-solving setting. This book was written with that thought in mind. Rather than going through the set of Z80 instructions and creating artificial uses, we have created problems and then discussed instructions that may be used to solve the problems. We also feel that it is much easier to learn to program by studying simple programs written by others than to strike out on one's own, armed with a mere bag full of instructions.

The book is centered around Radio Shack's T-BUG™* Monitor and Debugging Aid and eight small pages of information supplied in Radio Shack's T-BUG User Instruction Manual.

Equipment necessary to *use* the book, rather than merely *read* it, consists of:

1. A TRS-80 computer with 4K RAM
2. A video monitor
3. A cassette recorder
4. The T-BUG cassette

We assume that you have a knowledge of hexadecimal notation but have little or no background in machine language programming.

T-BUG is a machine language monitor designed to be used with the TRS-80 equipped with either Level I or Level II BASIC. The monitor allows you to enter machine language programs, examine and modify memory, and transfer control to

*T-BUG is a trademark of Radio Shack, a Division of Tandy Corp.

the program which you have entered.

T-BUG is provided on cassette tape by Radio Shack stores and is loaded into the TRS-80 by the use of the CLOAD command in Level I BASIC. If your TRS-80 is equipped with Level II BASIC, the SYSTEM command is used.

Level I BASIC was used in the early portions of this book until our Level II BASIC ROM arrived. However, all programs were verified and modified to show Level II use as well. There are some differences.

The locations of several subroutines used in the demonstration programs differ for the two levels of BASIC. The subroutines are also "packaged" differently. These differences will be pointed out when the subroutines are used.

All addresses, instructions and data are entered in hexadecimal notation (hereafter referred to as hex) and the video display uses the same notation when in T-BUG. Hex numbers, with their decimal equivalents, are given in Table 1, page 107.

This book has been written to introduce you to the use of T-BUG and is not intended to be a complete machine language text or to cover the entire Z80 instruction set. The pace is slow and is designed to provide you with information to use T-BUG immediately. Useful programs are used to demonstrate T-BUG's features with practical applications wherever possible.

SUMMARY OF T-BUG COMMANDS

 # M The MEMORY command is used to either modify or examine memory. A four-digit hexadecimal address must be typed to the right of M. The hex value of the contents of that memory location will then be displayed as:

 # M (address) (contents)

You type T-BUG responds

If the contents of the address are *not* to be changed, depress ENTER and the next higher memory address will be displayed along with its contents.

If the contents of the address are to be changed, you type in the correct value, and T-BUG automatically displays the next higher address along with its contents as:

 # M (address) (old contents) (new contents)

 # M (next address) (contents)

To exit the MEMORY function command mode:

Type: X

J The JUMP command transfers control to a specified memory location. The computer begins execution of the program from that location. This beginning memory location is entered as a four-digit hex number following the J. The specified location must contain the first byte of an operation code (opcode). The jump occurs immediately after entry of the address.

J (address)

Example: # J 4A00

Execution begins immediately after this last digit is typed.

B The BREAK command inserts a breakpoint at the address specified in the command. The address must be either the address of a single-byte instruction or the first byte of a multiple-byte instruction. When the user program reaches the specified address, the breakpoint terminates program execution and transfers control back to T-BUG. The format is:

B (address)

Example: # B 4A17

After the last digit of the address is typed, control is returned to T-BUG.

Only one breakpoint may be utilized at a time.

F This command is not named in the T-BUG manual so I will call it the FREE command. It "frees" the last breakpoint that was set (removes it and replaces the original instruction that the breakpoint replaced).

Example: # F

After the F is typed, control is returned to T-BUG.

G The GO command is used to continue execution of a program after a breakpoint has been used to halt execution and the F command used to remove the breakpoint.

Example:
```
# B 4A17
# J 4A00      Program is executed to the
  .           breakpoint. Corrections to the
  .           program may be made.
# F           Breakpoint is removed.
# G           Execution resumes at the point
              where the breakpoint was.
```

Execution resumes at the restored instruction with the registers automatically restored.

R The REGISTER command displays the contents of all the special and general purpose registers as shown on page 5 of the *T-BUG Users Instruction Manual* and on page 99 of this book. Register contents may be modified through the use of the MEMORY function. The appropriate RAM register storage locations are also found on page 5 of the *Users Instruction Manual.*

P The PUNCH command is used to store memory contents on a cassette tape. The command instructs the computer to write on the cassette all information between two user specified addresses. The starting address must be less than the ending address, and the cassette must be made ready by depressing the RECORD and PLAY buttons on the recorder. T-BUG controls the cassette tape motion (turning it on if the correct buttons have previously been set) and displays a # sign on the screen when the output has been completed.

The format for Level I BASIC is:

```
# P (start address) (end address)
```

Example:
```
# P 4A00 4A19
```
Execution begins when the last digit has been typed.

The format for Level II BASIC is:

P (start add.) (end add.) (entry add.) (file name)

Example: # P 4A00 4A45 4A20 GOBUGY

Execution begins when the 6th character has been typed.

If the file name contains less than 6 characters, the ENTER key must be pressed after the last character in the name.

L For Level I users, the LOAD command is used to load a program or data into the computer's memory from a cassette tape. It is assumed that the tape was prepared in the manner described under the PUNCH command. The data is loaded into the memory locations specified on the tape when punched. The cassette should be ready with the PLAY button depressed, and the cassette rewound to the correct starting point. The required user entry is:

L

As soon as the L is typed, the tape is set in motion and loaded. An asterisk (*) will blink as data is input, and the end of the loading function is signaled by the usual # sign appearing on the display.

If the tape is loaded improperly, T-BUG will display an **E** on the monitor to the right of the LOAD command. Try it again.

Level II users load cassettes via the SYSTEM command as described on page 42.

THE USE OF T-BUG MEMORY

The area of memory which may be used for your programs and data begins at location 4400 for Level I BASIC users and at location 4981 for Level II BASIC users. To avoid confusion, we have written all sample program solutions to start at location 4A00 which is within the user memory for both versions.

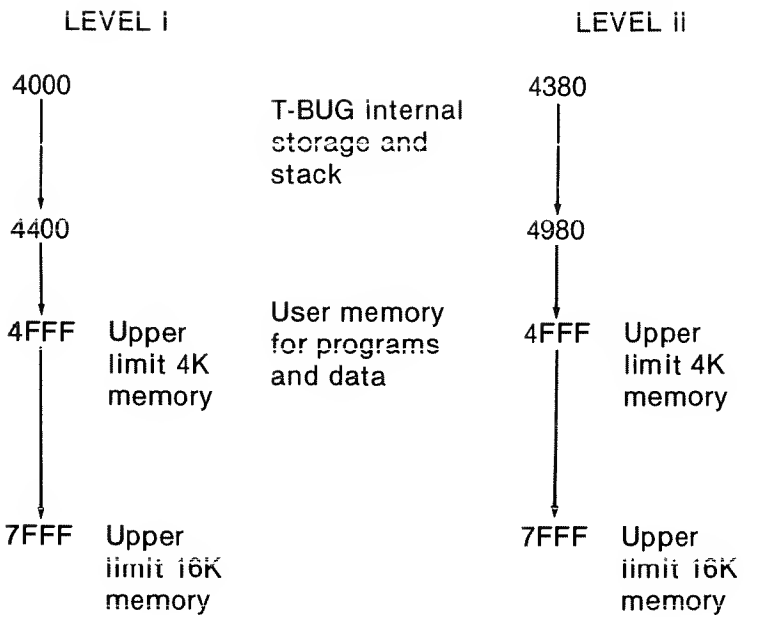


Figure 1.1 Memory Map

THE FIRST PROBLEM
Talking to the Computer

Before we try to do anything fancy with the computer, we should make sure that we can communicate with it.

Problem: Input some data from the keyboard and make it appear on the video display.

How to Reach a Solution: The computer must both scan the keyboard to see when a key has been struck and figure out which key it was. The stroke of each key puts out a distinctly coded electronic signal which is interpreted by the computer. The correct character for that key must then be displayed on the video screen. As each keystroke is displayed, the cursor (which keeps track of the displayed position on the screen) must be advanced one position to be ready for the next keystroke. We must also devise some way to terminate our program so that we can perform other tasks.

DEMONSTRATION PROGRAM NUMBER 1

Our first program is one of the most useful tools you will use with T-BUG. It allows you to input characters from the keyboard and display them on the screen. The cursor advances one position each time a keystroke is made. The program is shown on page 8 of the *T-BUG User Instruction Manual.* You will notice that it has a four-column format. We will add a name to each column and also put in two extra columns for the machine language instructions (called opcodes) and their

addresses.

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
LOOP			CALL	CHKIO	Scan keyboard
			JP	Z,LOOP	Go back if nothing there
MORCOD			CP	OD	Carriage return?
			JP	NZ,LOOP	If not, get another character

Figure 1.1 Table of Instructions – Program 1

Column I (LABEL) is optional. It is there to provide a reference point for jumps or to merely label different functional parts of a program. Column IV is the mnemonic (easy to remember) name for the instruction which is performed in conjunction with the operand which appears in Column V. The remarks in Column VI are also optional. They are used to explain what is "going on" in the program; they are not a part of the program itself. The remarks are useful to both the programmer and the program user. Columns II and III are for the "real" part of our machine language program. The address of each instruction (in hex notation) is placed in Column II. Column III is used for the hex representation of each mnemonic-operand pair. These values are found in a Z80 instruction code manual (also see Table II, pages 108 and 115). Each two-digit hex code occupies one memory location in the computer.

Line 1. CALL CHKIO is represented by:

LEVEL I	LEVEL II
CD (CALL)	CD (CALL)
40 (low-order address)	2B (low-order address)
0B (high-order address)	00 (high-order address)

The instruction CALL CHKIO is stored in three memory locations and is therefore called a three-byte instruction. Notice the difference in memory locations for the two versions of BASIC. If we place this instruction in our starting memory location, we would have:

LEVEL I		LEVEL II	
Address	Opcode	Address	Opcode
4A00	CD	4A00	CD
4A01	40	4A01	2B
4A02	0B	4A02	00

Notice the reverse order of CHKIO's address (which is actually OB40 or 002B).

Since three memory locations are used for this one instruction, we put them all on one line in Columns II and III of Figure I.I. The first line of the instruction table would look like this:

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
LOOP	4A00	CD 40 0B	CALL	CHKIO	Scan keyboard
LEVEL I					
LOOP	4A00	CD 2B 00	CALL	CHKIO	Scan keyboard
LEVEL II					

Although only 4A00 appears in the address column, we must remember that this instruction actually occupies three memory locations (4A00, 4A01 and 4A02).

Line 2. JP Z,LOOP is represented by:

CA (JP Z)
00 (low-order address of LOOP)
4A (high-order address of LOOP)

This is another three-byte instruction and will occupy the next three sequential memory locations. After the keyboard is scanned by CALL CHKIO, this instruction determines whether or not a key has been struck. If *not,* the instruction sends program execution back to scan the keyboard again. If a character *has* been struck, execution goes on to the next line of the program.

Our Level I table now looks like this:

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
LOOP	4A00	CD 40 0B	CALL	CHKIO	Scan keyboard
	4A03	CA 00 4A	JP	Z,LOOP	Go back if nothing there

The Level II version of these first two lines requires a more detailed approach. We must make sure that the accumulator is clear (0) before we scan for a keystroke. We do this by calling the keyboard scan routine, exclusive ØRing the accumulator with itself, and then calling the keyboard scan again to look for a keystroke. The equivalent of lines 1 and 2 in the Level I program become four lines for Level II.

Level II table becomes:

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
	4A00	CD 2B 00	CALL	CHKIO	Scan keyboard
	4A03	AF	XOR	A	Clear accumulator to zero
LOOP	4A04	CD 2B 00	CALL	CHKIO	Scan again
	4A07	CA 04 4A	JP	Z,LOOP	Go back if nothing there

LOOP has moved

Notice the extra call to CHKIO, and the insertion of the exclusive OR instruction at line 2. If the first two lines are omitted, an extraneous character, which we do not want, will appear due to some value which may be left in the accumulator.

The JP Z,LOOP instruction occupies three addresses in both versions of the program.

Next line. CP OD is represented by:

FE (CP or ComPare the value in the accumulator with the value which follows.)

OD (The hex value = 13 decimal. OD is the ASCII code for CARRIAGE RETURN.)

This two-byte instruction compares the ASCII code for the key which was struck to the value OD. If the values are the same, the Z (zero) flag is set to 1 (meaning the dif-

ference between the two values is zero). If the values are different, the Z flag is not set (it is sometimes called reset. The flag is reset to 0). Status flags, such as the zero flag, are merely individual bits in the status register which are set to a 1 or a 0 depending on the "status" of the computer after an instruction has been executed.

Our tables are growing. In the Level I version, CHKIO echos the keystroke on the display. However, an extra line must be added for the Level II version to display the keystroke.

Level I

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
LOOP	4A00	CD 40 0B	CALL	CHKIO	Scan keyboard
	4A03	CA 00 4A	JP	Z,LOOP	Go back if nothing there
MORCOD	4A06	FE 0D	CP	OD	Carriage return?

Level II

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
	4A00	CD 2B 00	CALL	CHKIO	Scan keyboard
	4A03	AF	XOR	A	Clear accumulator to zero
LOOP	4A04	CD 2B 00	CALL	CHKIO	Scan again
	4A07	CA 04 4A	JP	Z,LOOP	Go back if nothing
	4A0A	CD 33 00	CALL	OUTC	Display keystroke
MORCOD	4A0D	FE 0D	CP	OD	Carriage Return?

The next instruction demands a decision based on the comparison of the accumulator and the ASCII code for carriage return. If they are the same, the Z flag has been set to 1 as described earlier.

Next line. JP NZ,LOOP (Jump if the Z flag is *not* set) is represented by:

	C2 (JP NZ)	C2	
Level I	00 (low-order address of LOOP)	04	Level II
	4A (high-order address of LOOP)	4A	

If the zero flag is not set, execution of the program goes back to address 4A00 (4A04 for Level II). If it is set, the program goes on to address 4A0B (4A12 for Level II).

This short program is now complete. Our completed instruction tables for Level I and Level II are shown in Figures 1.2 and 1.3.

Level I

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
LOOP	4A00	CD 40 0B	CALL	CHKIO	Scan keyboard
	4A03	CA 00 4A	JP	Z,LOOP	Back if nothing
MORCOD	4A06	FE 0D	CP	OD	Carriage Return?
	4A08	C2 00 4A	JP	NZ,LOOP	No, get next one

Figure 1.2 Final Instruction Table—Program 1

Level II

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
	4A00	CD 2B 00	CALL	CHKIO	Scan keyboard
	4A03	AF	XOR	A	Clear accumulator to zero
LOOP	4A04	CD 2B 00	CALL	CHKIO	Scan again
	4A07	CA 04 4A	JP	Z,LOOP	Back if nothing
	4A0A	CD 33 00	CALL	OUTC	Display it
MORCOD	4A0D	FE 0D	CP	OD	Carriage Return?
	4A0F	C2 04 4A	JP	NZ,LOOP	No, get next one

Figure 1.3 Final Instruction Table—Program 1

Program Review

The program scans the keyboard until a key is struck (lines 1 and 2 for Level I, lines 3 and 4 for Level II). When this happens, the character for that key is echoed on the video display. The computer then compares the ASCII code for the typed character with OD, the ASCII code for a carriage return.

If the two values do not match, execution of the program is returned to scan the keyboard for another character. What will happen if a carriage return has been struck?

We have made no provision for this possibility. For the time being, let's not worry about it. We'll use a breakpoint after we have entered the program. When a carriage return is entered, the program will encounter the breakpoint and transfer control back to the T-BUG monitor.

After the program is entered we will be able to type in any message and see it displayed on the video screen. We will be communicating with the computer. Note, however, that the computer is merely repeating everything which we type. For the computer to understand what we are saying will require more programming.

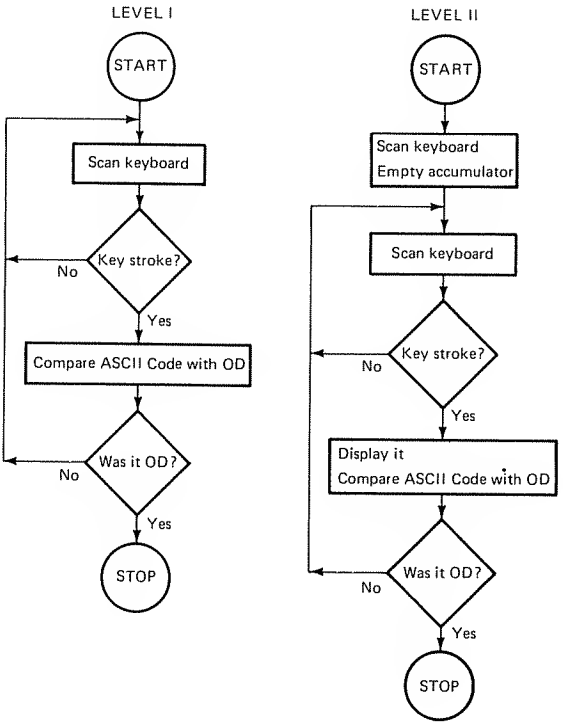


Figure 1.4 Flowchart of Program 1

LOADING T-BUG FROM LEVEL I BASIC

With your TRS-80 and recorder connected, apply power. The computer will be ready for you to program in BASIC language as the video display responds with:

```
READY
>_
```

The > is a signal that the computer is in the BASIC ROM, and the _ shows the present cursor position. Whatever you input from the keyboard will be displayed at the cursor position.

Now it's your turn to load the T-BUG cassette. Be sure the cassette is in the recorder and rewound to the beginning of the tape.

1. Type in: CLOAD. (Do *not* press the ENTER key yet!)
2. Press the PLAY button on the recorder. Nothing should happen yet because the computer controls the movement of the cassette.
3. Now, press the ENTER key on the TRS-80 keyboard. The cassette should start. After a short time, two asterisks appear, and the right one blinks. This is a visual sign that the loading is in progress.
4. When T-BUG has finished loading into the computer, the # sign appears on the screen, and the recorder is stopped.

```
#
```

5. Press the STOP button on the recorder. The # sign is an indication that control has been passed from the BASIC ROM to the T-BUG monitor. T-BUG is now ready for the program to be entered.

LOADING T-BUG FROM LEVEL II BASIC

With your TRS-80 recorder connected, apply power. The computer will respond: **MEMORY SIZE?** _.

1. You press ENTER .
2. The computer responds:

```
MEMORY SIZE?
RADIO SHACK LEVEL II BASIC
READY
>_
```

3. You type: SYSTEM.
4. Press ENTER .
5. The computer responds:

```
>SYSTEM
*?_
```

6. You type: TBUG. Do *not* press ENTER yet.
7. Press the PLAY button on your recorder. Nothing should happen yet because the computer has control of the recorder.
8. Now press ENTER on the TRS-80 keyboard. The recorder should start. After a short time, two asterisks appear, and the right one blinks.
9. When T-BUG has finished loading, the computer displays another *?__ and the recorder is stopped.
10. Press the STOP button on your recorder.
11. Now type: / and press ENTER .
12. The # sign appears on the screen, indicating that control is now in the T-BUG monitor. T-BUG is now ready for the program to be entered.

```
#
```

NOW LET'S USE T-BUG

The # sign is on the screen.

1. *To examine* or *alter memory:* Type: M

```
# M          M for
             "Memory"
```

2. Now type in the beginning address: 4A00

```
# M 4A00 XX  Two hex digits will
             appear at XX. That
             value is in 4A00 now.
```

3. Our program starts with the value CD so we type: CD

```
# M 4A00 XX CD    The CD has replaced
4A01 YY           the old value XX.
```

The next address appears. *Voila!*

Two hex digits appear after the address.

4. Next in our program is the CALL *address*. Type in: 40 for Level I; or 2B for Level II.

```
# M 4A00 XX CD
4A01 YY 40        New value
4A02 XX
```

There's the next address and its contents

```
# M 4A00 XX CD
4A01 YY 2B
4A02 XX
```

5. Next type: 0B for Level I; or 00 for Level II.

```
# M 4A00 XX CD
4A01 YY 40
4A02 XX 0B        New value
4A03 YY
```

Next address

```
# M 4A00 XX CD
4A01 YY 2B
4A02 XX 00
4A03 YY
```

6, 7, 8, etc. This process continues. All you have to do is type in the opcode values from our table (page 12). T-BUG and the computer do the rest.

We finally type our last step: 4A and the program is completely entered as is shown.

The complete entry sequence for both Level I and II is shown in Figure 1.4.

LEVEL I		LEVEL II
# M 4A00 XX CD		# M 4A00 XX CD
4A01 YY 40		4A01 YY 2B
4A02 XX 0B		4A02 XX 00
4A03 YY CA	The last values	4A03 YY AF
4A04 XX 00	in the rows are	4A04 XX CD
4A05 YY 4A	the codes for	4A05 YY 2B
4A06 XX FE	our program.	4A06 XX 00

```
4A07 YY 0D
4A08 XX C2
4A09 YY 00
4A0A XX 4A
4A0B YY
```

When the program has been entirely entered you will see only the last 16 lines on the screen.

```
4A07 YY CA
4A08 XX 04
4A09 YY 4A
4A0A XX CD
4A0B YY 33
4A0C XX 00
4A0D YY FE
4A0E XX 0D
4A0F YY C2
4A10 XX 04
4A11 YY 4A
4A12 XX
```

Figure 1.4 Program Number 1 as Entered

TO GET OUT OF THE MEMORY EXAMINE/ALTER MODE

Type: X

The display will roll up one line and the # sign will appear at the bottom of the program.

To finish the program we will use the breakpoint we mentioned earlier. After the number sign that appeared when we exited the memory mode, type:

B4A0B for Level I or B4A12 for Level II.

"B" for breakpoint — Its address

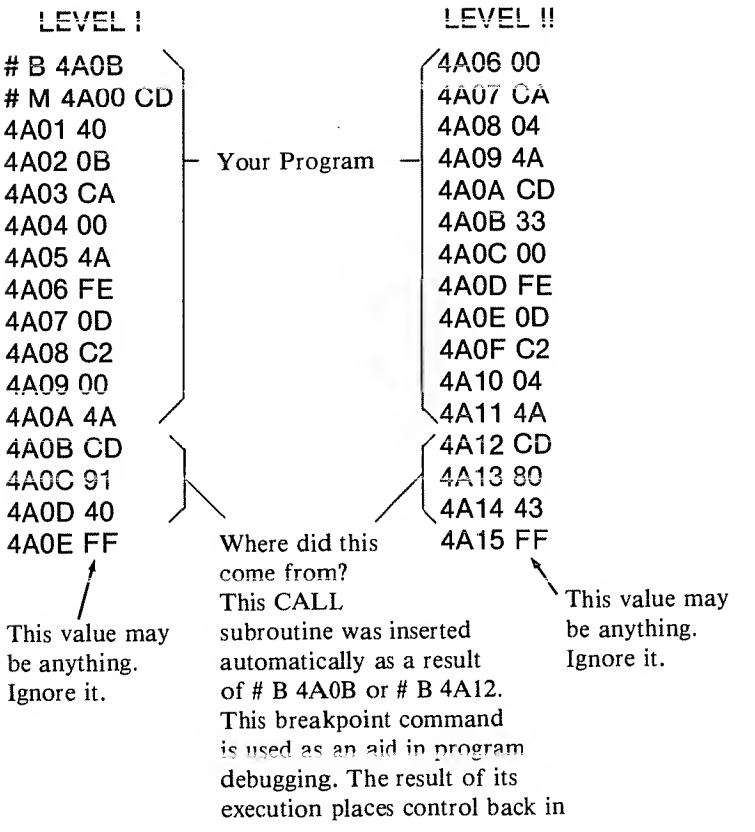
EXAMINE MEMORY

After entering a program it is always a good idea to check and make sure you entered it correctly. It is easy to make mistakes when working in hex. To examine what you have in memory is quite simple. All you have to do is type an M followed by the first address that you want to examine. The computer then prints that address and the value which that address location holds. To view the next sequential address, just press the ENTER key. To examine a string of sequential addresses, such as a previously entered program, just keep pressing the ENTER key. Each stroke of the ENTER key provides a new address and its contents.

When you entered the program breakpoint in the program that we are working with, the # sign was displayed on the screen. Following that # sign:

1. Type M 4A00.
2. Press the ENTER key.
3. Repeat step 2 until the complete program has been checked for correct entry.
4. If any corrections are to be made, enter the correction immediately following the value that is in error. After the correction is made, you do not have to press the ENTER key as in step 2. The computer takes care of that automatically when a correction is made.

When you have completed the examination and correction of your program, you should see the following sequence on your screen:



the T-BUG command mode so that you may take any action that you desire.

We now have the program entered correctly. To get out of the MEMORY EXAMINE mode, type: X.

Our # sign appears again. T-BUG is ready for another command, and we are ready to RUN our program.

RUNNING THE PROGRAM

All you need to do to run the program is type:

J (for JUMP to) followed by 4A00

Now press the CLEAR key on the keyboard. If this doesn't clear the screen, continually press the down arrow ↓ key until the screen is cleared.

Now type the following message on the keyboard. (Don't use the carriage return at the end of a line.)

```
DEMONSTRATION PROGRAM NUMBER 1 FOR T-BUG. ANYTHING YOU NOW TYPE
WILL APPEAR ON THE SCREEN. WHAT WILL HAPPEN IF YOU PRESS THE CA
RRIAGE RETURN (THAT'S THE ENTER KEY ON THE TRS-80)? TRY IT
```

What happened???? I thought I told you not to hit the carriage return. The program was terminated, and the computer jumped back to the T-BUG monitor.

Here's what I see on my display:

```
#          OGRAM NUMBER 1 FOR T-BUG. ANYTHING YOU NOW TYPE
           HE SCREEN. WHAT WILL HAPPEN IF YOU PRESS THE CA
           HAT'S THE ENTER KEY ON THE TRS-80)? TRY IT.
```

This gets blanked out, and our friend, the # sign, appeared. That means we're back in T-BUG COMMAND mode.

The way our program is now written, every time we hit the carriage return key, the breakpoint sends us back to the monitor. This is ridiculous. We must be able to use the carriage return in the regular way. Can't we change something so that the program won't jump back to T-BUG on every carriage return?

Why not make the comparison to something that we are not likely to use when typing text? Let's try the @ symbol. The

ASCII code for @ is 40 (hex). Therefore, we must replace the compare instruction (FE 0D) with FE 40. We can do this easily with T-BUG using the ALTER MEMORY mode.

ALTER MEMORY

Our # sign is already on the screen as a result of executing the breakpoint. All we have to do is type M4A07 for Level I or M4A0E for Level II after the # sign. The display shows:

LEVEL I

```
# M 4A07 OD
```

LEVEL II

```
# M 4A0E OD
```

Now type: 40. The display shows:

LEVEL I

```
# M 4A07 OD 40
4A08 C2
```

LEVEL II

```
# M 4A0E OD 40
4A0F C2
```

Now type: X. The display shows:

LEVEL I

```
# M 4A07 OD 40
4A08 C2
#
```

LEVEL II

```
# M 4A0E OD 40
4A0F C2
#
```

Now run the program again by typing J4A00 and press CLEAR (or ↓ pressed several times) to clear the screen.

You may now use the carriage return or whatever else you want to type (except the @ sign, of course). When you are tired of typing messages, try a few graphic designs. If you've had enough, then hit the @ key to get back to the monitor.

End of Problem 1

THE SECOND PROBLEM
Displaying Data from Memory

You learned to talk to the TRS-80 through T-BUG by solving Problem 1. However, the program merely echoed the characters that you typed. Now let's go one step farther.

Problem: Retrieve information that has been previously stored in memory, and display the information on the video screen.

How to Reach a Solution: Write a program which will display characters from ASCII coded information previously stored. We will show two separate solutions.

A. Place each character in the accumulator, display it immediately, then go on to the next character.
B. Recall each character from the memory location where it has been previously stored.

Each solution will terminate with instructions similar to that used in our first program.

SOLUTION TO PROBLEM 2A

This program is also adapted from a suggested subroutine shown on page 8 of the *T-BUG Users Manual.* This program for Level I BASIC uses two instructions, LOAD ACCUMULATOR IMMEDIATE and RESTART 2. The program for Level II is similar. However, it uses a CALL instruction to output the character instead of the RESTART used in the Level I version.

The Z80 mnemonic for LOAD ACCUMULATOR IM-

MEDIATE is: LD.

The operand is: reg,data.

The opcode for this instruction (in binary notation) is:

00 XXX 110 yy

where the XXX bits are selected from the table below according to the register that is to be used. The symbols, yy, are replaced by the hex value of the data that is to be loaded.

REGISTER TABLE

000 = B
001 = C
010 = D
011 = E
100 = H
101 = L
111 = A

In our particular case, we are using register A, the accumulator.

Figure 2.1 Register Code Table

Thus the instruction in our problem becomes:

00 111 110 yy or

(in hex) 3 E yy

The Z80 mnemonic for RESTART is: RST.

The operand is: n.

The opcode is: 11 XXX 111 where XXX is replaced by the binary value of n. In our case, we are using the value 010 for n, and the instruction becomes:

11 010 111 or

(in hex) D 7

Level II users will call a subroutine named OUTC (output a character) which resides in memory beginning at location 0033. It is a three-byte instruction:

CD the opcode
33 the low-order address byte
00 the high-order address byte

Tables similar to those used for Program 1 would start with the following two lines:

Level I

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	3E 50	LD	A,50	Load ASCII code
	4A02	D7	RST	2	Display it

Level II

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	3E 50	LD	A,50	Load ASCII code
	4A02	CD 33 00	CALL	OUTC	Display it

This part of the program would place a **P** on the video screen. The hex number, 50, is the ASCII value for the letter P. The cursor would be advanced one position ready for the next keystroke. The two instructions would be repeated over and over by our program until our message on the screen was complete. Of course, we would insert different ASCII values each time to make the message meaningful.

Let's write a program which will display:

PROGRAM # 2

On the following two pages you will find tables for part of the new program. One is for Level I users, and the other is for Level II users. See if you can fill in the blanks in the address, opcode, and operand columns for your particular version of BASIC. You'll need an ASCII table and the Z80 instruction set. These can be found in the appendix at the back of this book.

Level I

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	3E 50	LD	A,50	Load ASCII P
	4A02	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII R
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII O
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII G
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII R
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII A
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII M
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII carriage return
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII #
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII 2
	____	D7	RST	2	Display it
	____	__ __	LD	A,__	Load ASCII carriage return
	____	D7	RST	2	Display it

Figure 2.2 Worksheet for Program 2A

Level II

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	3E 50	LD	A,50	Load ASCII P
	4A02	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII R
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII O
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII G
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII R
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII A
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII M
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII carriage return
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII #
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII 2
	____	CD 33 00	CALL	OUTC	Display it
	____	__ __	LD	A,__	Load ASCII carriage return
	____	CD 33 00	CALL	OUTC	Display it

Figure 2.3 Worksheet for Program 2A

We will make use of Program 1 as the concluding part of Program 2A. The complete Machine Language Program to be entered for Level I is:

ADDRESS	CODE	
4A00	3E	
4A01	50	P
4A02	D7	
4A03	3E	
4A04	52	R
4A05	D7	
4A06	3E	
4A07	4F	O
4A08	D7	
4A09	3E	
4A0A	47	G
4A0B	D7	
4A0C	3E	
4A0D	52	R
4A0E	D7	
4A0F	3E	
4A10	41	A
4A11	D7	
4A12	3E	
4A13	4D	M
4A14	D7	
4A15	3E	
4A16	0D	carriage return
4A17	D7	
4A18	3E	
4A19	23	#
4A1A	D7	
4A1B	3E	
4A1C	32	2
4A1D	D7	
4A1E	3E	
4A1F	0D	carriage return
4A20	D7	
4A21	CD	
4A22	40	
4A23	0B	

Your answers on page 24 should agree with these.

4A24	CA
4A25	21
4A26	4A
4A27	FE
4A28	40
4A29	C2
4A2A	21
4A2B	4A
4A2C	CD
4A2D	91
4A2E	40

All this is from program 1 with appropriate jump addresses changed.

Program 2A for Level II is a little longer.

ADDRESS	CODE	
4A00	3E	
4A01	50	P
4A02	CD	
4A03	33	
4A04	00	
4A05	3E	
4A06	52	R
4A07	CD	
4A08	33	
4A09	00	
4A0A	3E	
4A0B	4F	O
4A0C	CD	
4A0D	33	
4A0E	00	
4A0F	3E	
4A10	47	G
4A11	CD	
4A12	33	
4A13	00	
4A14	3E	
4A15	52	R
4A16	CD	
4A17	33	
4A18	00	
4A19	3E	

Your answers on page 25 should agree with these.

4A1A	41	A
4A1B	CD	
4A1C	33	
4A1D	00	
4A1E	3E	
4A1F	4D	M
4A20	CD	
4A21	33	
4A22	00	
4A23	3E	
4A24	0D	carriage return
4A25	CD	
4A26	33	
4A27	00	
4A28	3E	
4A29	23	#
4A2A	CD	
4A2B	33	
4A2C	00	
4A2D	3E	
4A2E	32	2
4A2F	CD	
4A30	33	
4A31	00	
4A32	3E	
4A33	0D	carriage return
4A34	CD	
4A35	33	
4A36	00	
4A37	CD	
4A38	2B	
4A39	00	
4A3A	AF	
4A3B	CD	
4A3C	2B	
4A3D	00	
4A3E	CA	
4A3F	3B	
4A40	4A	
4A41	CD	
4A42	33	
4A43	00	

All this is from Program 1 with appropriate jump addresses changed.

4A44	FE
4A45	40
4A46	C2
4A47	3B
4A48	4A
4A49	CD
4A4A	80
4A4B	43

LOADING PROGRAM 2A

Load T-BUG in the manner described for Program 1. Then you can load the program.

1. When the # sign appears, type: M 4A00

```
# M 4A00 XX
```

2. Enter the machine language code for each appropriate address. Remember, T-BUG automatically gets the next address after each entry that you make. If you make a typing error, you can correct it later. Start your entries as follows:

Level I

```
# M 4A00 XX 3E
4A01 YY 50
4A02 XX D7
4A03 YY 3E
4A04 XX 52
   .
   .
  etc.
```

Level II

```
# M 4A00 XX 3E
4A01 YY 50
4A02 XX CD
4A03 YY 33
4A04 XX 00
   .
   .
  etc.
```

3. When the program has been completely entered, type: X. This exits the MEMORY mode and returns you to the COMMAND mode with the # sign displayed.

4. Now type: M 4A00 to examine your entries. If an entry is correct, press ENTER to view the next entry. If an entry is incorrect, type the correct value for that address. T-BUG will enter the new value and display the next one.

5. When all entries have been verified, type: X and the # sign appears again.
6. A new way to clear the screen is shown. At the bottom of the screen are your last entries:

Level I

```
4A2C CD
4A2D 91
4A2E 40
#
```

Level II

```
4A49 CD
4A4A 80
4A4B 43
#
```

Following the # sign, type: B 4B00. Then type: J 4B00. The screen will clear (at least the left 16 columns) and the number sign will appear at the upper left.

```
#
```

7. To RUN the program, type: J 4A00 and there it is:

```
PROGRAM
# 2
```

You are now looping in that portion of the program which was taken from Program 1. Type in any message that you want, and it will appear on the screen. Feel free to get rid of any frustrations you are having with machine language programming.

PROGRAM 2B

Now let's look at the second method which displays ASCII data stored in memory. We'll put the ASCII data in memory first, and then load the data into the accumulator from memory via our program. The program will be much shorter and more efficient.

Two registers (B and C) will be used to "point" to the correct memory location as data is required by the program. Registers, such as B and C, are special memory locations used by many instructions in the Z80 set. This program will demonstrate one use of these registers.

Here are the ASCII codes that we used in Program 2A along with the memory locations where they will be stored.

ADDRESS	CODE	CHARACTER
4B00	50	P
4B01	52	R
4B02	4F	O
4B03	47	G
4B04	52	R
4B05	41	A
4B06	4D	M
4B07	0D	carriage return
4B08	23	#
4B09	32	2
4B0A	0D	carriage return

TO LOAD THE DATA

Access the monitor in the usual manner so that the # sign is showing.

1. Type: M 4B00

```
# M 4B00 XX
```

2. Type the ASCII code for P: 50

```
# M 4B00 XX 50
4B01 YY
```

3. Type the ASCII code for R: 52

```
# M 4B00 XX 50
4B01 YY 52
4B02 XX
```

4. Type the ASCII code for O: 4F

```
# M 4B00 XX 50
4B01 YY 52
4B02 XX 4F
4B03 YY
```

Continue with:

5. Type the ASCII code for G: 47
6. Type the ASCII code for R: 52
7. Type the ASCII code for A: 41
8. Type the ASCII code for M: 4D
9. Type the ASCII code for carriage return: 0D
10. Type the ASCII code for #: 23
11. Type the ASCII code for 2: 32
12. Type the ASCII code for carriage return: 0D

Once the data has been entered, we are ready for the program. Before entering the program and using it, you should have a general idea of how it works.

HOW THE PROGRAM WORKS

As mentioned earlier, we will use the B and C registers to tell the computer where to find the data. We need two registers because each one of them holds only one byte of data. Addresses are two bytes long—hence, two registers.

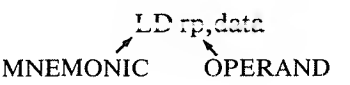
Our first byte of data was placed in address 4B00. Therefore, when the program starts, the BC register pair should be "pointing" to address 4B00. To do this, we load 4B into register B and 00 into register C.

The program loads the accumulator from the address held in the register pair, BC. The character represented by the ASCII code (which is in the memory location pointed to by the BC register pair) is then displayed on the video screen.

The value in register C is then incremented by one. Now the BC register pair points to memory location 4B01. The value in register C is then tested to see if all the characters have been displayed. If *not*, we repeat the process using the value in the BC register pair to select each new character.

The program introduces several new instructions. Three of them are LOAD instructions. We used LOAD ACCUMULATOR IMMEDIATE in Program 2A. Each of these LOAD instructions is used for a different purpose.

LOAD a Register Pair with 16 Bits of Data:



The opcode is made up of the following binary bits which have been grouped for easy hex conversion.

00XX 0001 where XX is the code for a particular register pair and is selected from Figure 2.4.

CODE	REGISTER PAIR
00	BC
01	DE
10	HL
11	Stack Pointer

Figure 2.4 Register Pair Codes—Data

We will use the BC register pair, so the opcode is:

0000 0001 binary or

01 hex

Our instruction is thus:

01 00 4B
LD BC,4B00

since we want the BC register pair to "point" to address 4B00.

LOAD Accumulator from Memory Addressed by a Register Pair

LD A,(rp)

MNEMONIC OPERAND

The opcode is determined by the register pair to be used as follows:

000X 1010 binary where X is selected from Figure 2.5 below. Only the register pairs BC and DE are used with this instruction.

CODE	REGISTER PAIR
0	BC
1	DE

Figure 2.5 Register Pair Codes—Addresses

Since we are using BC in our program, the opcode is:

0000 1010 binary or

0A hex

Our instruction becomes:

```
OA
LD A,(BC)
```

The parentheses denote "the contents of" or "from the memory pointed to by (XX).

This one-byte instruction copies the value contained in the memory location pointed to by the register pair BC into the accumulator. A long mouthful, but a very useful instruction.

LOAD One Register (Destination) from Another Register (Source)

```
      LD dst,src
MNEMONIC  OPERAND
```

The opcode is determined by the registers used as follows:

01 ddd sss binary where the values chosen for ddd and sss are selected from the table given in Figure 2.1.

We will be copying data from the source register (C) into the destination register (A). Therefore, our opcode is:

01 111 001 binary or

0111 1001 binary for easier hex conversion

79 hex

Our instruction then, is: 79
LD A,C

INCREMENT a Register

INC reg
MNEMONIC OPERAND

The opcode for this new instruction is also determined from Figure 2.1 according to the register to be used.

The opcode is: 00 XXX 100

Since we are using register C, we want:

00 001 100 binary or

0000 1100 binary for easy hex conversion

Our instruction in hex is: OC
INC C

This instruction increases the current value in register C by one.

Program 2B for Level I Users

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	01 00 4B	LD	BC,4B00	Load BC regs.
LOOP	4A03	OA	LD	A,(BC)	Load A from memory
	4A04	D7	RST	2	Display it
	4A05	OC	INC	C	Count up 1
	4A06	79	LD	A,C	Get C's value
	4A07	FE 0B	CP	0B	Compare to 0B
	4A09	C2 03 4A	JP	NZ,LOOP	If not 0B, go back
HOLD	4A0C	CD 40 0B	CALL	CHKIO	Scan keyboard
	4A0F	CA 0C 4A	JP	Z,HOLD	Check for keystroke
	4A12	FE 40	CP	40	Was it @
	4A14	C2 0C 4A	JP	NZ,HOLD	If not, go back
	4A16	CD 91 40	CALL	MON	Go to T-BUG

Program 2B for Level II BASIC Users

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	01 00 4B	LD	BC,4B00	Load pointer
LOOP	4A03	0A	LD	A,(BC)	Load data
	4A04	CD 33 00	CALL	OUTC	Display it
	4A07	0C	INC	C	Pointer +1
	4A08	79	LD	A,C	Load C
	4A09	FE 0B	CP	0B	See if done
	4A0B	C2 03 4A	JP	NZ,LOOP	Go back if not
HOLD	4A0E	CD 2B 00	CALL	CHKIO	Scan keyboard
	4A11	CA 0E 4A	JP	Z,HOLD	Try again if nothing
	4A14	FE 40	CP	40	Was it @?
	4A16	C2 0E 4A	JP	NZ,HOLD	Go back if not
	4A19	CD 80 43	CALL	MON	Done

TO LOAD PROGRAM 2B

We will assume that you have loaded the data into memory locations 4B00 through 4B0A and are back in the monitor with the # sign on the display.

Type: M 4A00
01
00
4B
CA
. .
etc. until the program has been entered.

Level I

```
# M 4A00 XX 01
4A01 YY 00
4A02 XX 4B
4A03 YY 0A
4A04 XX D7
4A05 YY 0C
4A06 XX 79
4A07 YY FE
4A08 XX 0B
```

The last two characters are your entries. XXs and YYs are values that were in memory prior to your entries.

Level II

```
# M 4A00 XX 01
4A01 YY 00
4A02 XX 4B
4A03 YY 0A
4A04 XX CD
4A05 YY 33
4A06 XX 00
4A07 YY 0C
4A08 XX 79
```

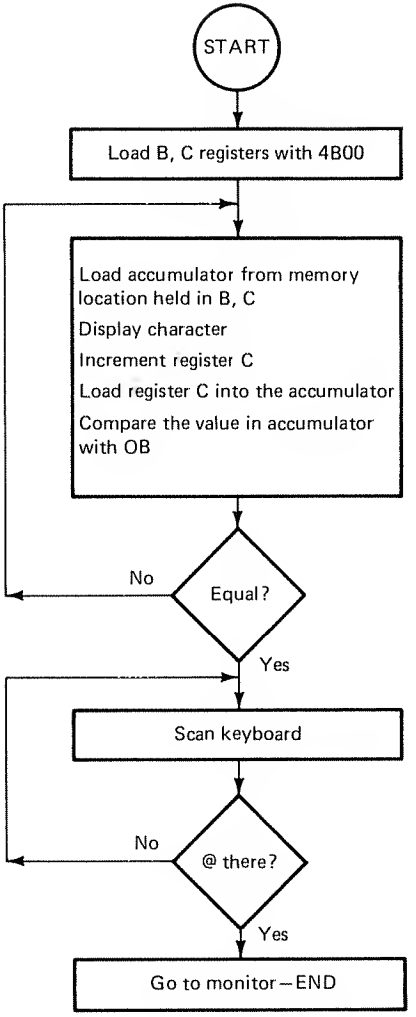


Figure 2.6 Flowchart for Program 2B

```
4A09 YY C2
4A0A XX 03
4A0B YY 4A
4A0C XX CD
4A0D YY 40
4A0E XX 0B
4A0F YY CA
```

```
4A09 YY FE
4A0A XX 0B
4A0B YY C2
4A0C XX 03
4A0D YY 4A
4A0E XX CD
4A0F YY 2B
```

```
4A10 XX 0C
4A11 YY 4A
4A12 XX FE
4A13 YY 40
4A14 XX C2
4A15 YY 0C
4A16 XX 4A
4A17 YY CD
4A18 XX 91
4A19 YY 40
4A1A XX
```

```
4A10 XX 00
4A11 YY CA
4A12 XX 0E
4A13 YY 4A
4A14 XX FE
4A15 YY 40
4A16 XX C2
4A17 YY 0E
4A18 XX 4A
4A19 YY CD
4A1A XX 80
4A1B YY 43
4A1C XX
```

When you have finished loading the program,
Type: X
Then type: M 4A00 to step through your program to check the entries.

TO RUN THE PROGRAM

Having entered the data at memory locations 4B00 through 4B0A and the program at memory locations 4A00 through the appropriate final location, you are now ready to try it out. Type: J 4A00 and the display responds:

```
PROGRAM
# 2
```

Don't forget that after the message is displayed the program is in a loop and will only exit the loop if you press the @ key. You may wish to change the data which is being displayed. **PROGRAM # 2** is not a very interesting message.

TO ALTER THE DISPLAYED MESSAGE

1. Load ASCII data for whatever message you desire in memory locations 4B00-4B??.
2. Change the value at location 4A08 (or 4A0A for Level II) to the number of characters that you wish to display (hex notation).
3. Rerun the program with a: J 4A00.

THE THIRD PROBLEM
Using the Recorder

Sooner or later, your programs are going to be so good that you will want to save them for future use. They will also grow to such a size that it will be tedious to load them from the keyboard every time that you want to use them.

Problem: Saving programs for later use and getting them quickly and easily into the computer.

How to Reach a Solution: Save the program on a tape cassette and use the recorder to reload the program when desired. Two methods to produce this result will be shown for Level I BASIC users and one method for Level II. The first method for Level I makes use of the PUNCH and LOAD commands from the monitor. The second method makes use of T-BUG subroutines CTON, CSAVEO and CLOADO. The PUNCH, LOAD method is used from the COMMAND mode; and the CTON, CSAVEO, CLOADO method can be incorporated into a program. The other method, for Level II, uses the PUNCH command, with a special format, for saving programs, and the SYSTEM command for loading.

SAVING A PROGRAM ON A CASSETTE—LEVEL I

As noted in the T-BUG manual, the PUNCH command provides a 128-byte leader, a synchronization code, the starting and ending addresses, the data, and a one-byte check sum. It records

all the information between the two user provided addresses.

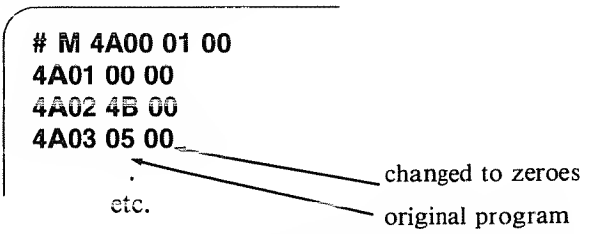
Assuming you are in the command mode, and the usual # sign shows on the display:

1. Enter Program #2B from the keyboard.
2. Type: X (to exit the MEMORY mode).
3. The # sign appears again.
4. Press the PLAY and RECORD buttons on your recorder. T-BUG has control of the tape movement so nothing happens yet.
5. Type: P 4A00 4B0B and the tape motion should start, and the program will be recorded.
6. Another # sign will appear on the display to signal that the program recording has finished.

The program is saved (you hope). It is advisable to record a program at least twice in case of problems. To do this, repeat steps 4 and 5. Don't forget to *make a note of the index settings* of the recording for future use. There is no way to name files of your recordings via T-BUG using Level I. Therefore, you must keep written records (i.e., name of program, what it does, how to use it, where to find it on tape, etc.).

ENTERING THE PROGRAM FROM CASSETTE—LEVEL I

Pretend now that it is a few days later, and you want to run the program again. Rather than enter the program from the keyboard, let's use the tape that you just saved. To make sure that the recording works, use the MEMORY command to put zeroes in some of the memory locations.



Now, if the program really loads from the tape, we'll know that we are not just seeing data left from the original program.

Type: X to get the # sign back.

1. Rewind the cassette to the index number where the start of your recorded program exists.
2. Press the PLAY button on the recorder.

3. Type: L (The tape motion should start and two asterisks appear at the top right corner of the display. The right one should blink.)
4. When the program has been loaded, the screen will clear and the # sign will appear again. (An **E** appearing to the right of the LOAD command means an error in loading has occurred. Try again.)

SAVING A PROGRAM ON A CASSETTE—LEVEL II

A note on page 9 of the T-BUG manual describes the use of the PUNCH command for Level II BASIC users. It is similar to that of Level I except for some additional entries. Four entries must follow the letter **P**:

1. The starting address
2. The ending address
3. The program's entry point (which is sometimes different from the starting address)
4. The program's file name (up to 6 characters)

Assuming that you are in the command mode, and the # sign shows on the display:

1. Enter Program 2B (page 00) from the keyboard (don't forget the data—memory locations 4B00-4B0B).
2. Type: X (to exit the MEMORY mode).
3. The # sign appears.
4. Press the PLAY and RECORD buttons on your recorder. T-BUG has control of the tape movement so nothing should happen yet.
5. Type: P 4A00 4B0B 4A00 DEMOTB

start add.	end add.	entry point	file name

   The tape motion should start immediately if your file name is 6 characters long. If it is less than 6, you must press the ENTER key.
6. When the recording is complete, another # sign will appear on the display as a prompt.
7. To make an additional copy of the program, repeat steps 4, 5 and 6. The file name provides you with a method of locating this particular program on your cassette when you are ready to load it from tape.

ENTERING THE PROGRAM FROM CASSETTE—LEVEL II

Programs may be entered in the same way that T-BUG is loaded for Level II—by means of the SYSTEM command. Let's use the demonstration tape that we have just recorded (DEMOTB) to illustrate the procedure.

Assume that your computer is off. The first thing to do is load T-BUG.

1. Turn on the computer and the display shows:

```
MEMORY SIZE? _
```

2. Press ENTER .
3. The computer responds:

```
MEMORY SIZE?
RADIO SHACK LEVEL II BASIC
READY
> _
```

4. Type: SYSTEM and press ENTER .
5. The computer responds:

```
MEMORY SIZE?
RADIO SHACK LEVEL II BASIC
READY
> SYSTEM
*?_
```

6. Type: TBUG (but do *not* press ENTER yet).
7. Press PLAY on the recorder (nothing should happen yet).
8. Now press ENTER on the TRS-80 keyboard. The recorder starts, and soon the two asterisks appear and the right one blinks.
9. When the loading is completed, the display shows another *? _ and the recorder is stopped. Press the STOP button on your recorder.
10. Type the file name: DEMOTB (but do *not* press ENTER yet).
11. Press PLAY on the recorder (nothing should happen yet).

12. Now press ENTER on the TRS-80 keyboard. The recorder should start. After a short time, the two asterisks appear, and the right one blinks.
13. When the loading is completed, the display shows another ***?** _ and the recorder is stopped. Press the STOP button on your recorder.

```
MEMORY SIZE?
RADIO SHACK LEVEL II BASIC
READY
> SYSTEM
*? TBUG
*? DEMOTB
*? _
```

14. Type: / and press ENTER , and the program will automatically run (starting at the entry point named in your tape).

A SECOND METHOD FOR LEVEL I USERS

The Level I T-BUG user can make use of some handy subroutines described on pages 8 and 9 of the T-BUG manual. They are CTON, which turns the cassette on, CSAVEO, which saves a program, data, or both; and CLOADO, which loads a program.

By modifying Program 2B in the following way, we can automatically save our program. We will also change the data, as shown on the next page. This *added* portion of the program will save the data used in our program *if the recorder is set for recording*. It works like this:

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4A17	CD	CALL	CTON	Start the recorder
4A18	E9			
4A19	0F			
4A1A	26	LD	H,4A	Load starting address
4A1B	4A			(4A00) in the HL
4A1C	2E	LD	L,00	register pair
4A1D	00			
4A1E	16	LD	D,4A	Load ending address
4A1F	4A			of the data (plus 1) in

4A20 4A21	1E 48	LD	E,48	the DE register pair
4A22 4A23 4A24	CD 4B 0F	CALL	CSAVEO	Records data starting from the memory location stored in HL and ending in the location named by DE
4A25 4A26 4A27	CD 91 40	CALL	MON	When the program has been saved go back to T-BUG

NOTE: The extra location called for by DE is used to store a check sum for error detection.

Fill in the ASCII codes for the data to be used in our modified program. Then enter the data via the MEMORY command.

ADDRESS	CHARACTER	CODE
4A30	D	_____
4A31	E	_____
4A32	M	_____
4A33	O	_____
4A34	space	_____
4A35	O	_____
4A36	F	_____
4A37	space	_____
4A38	C	_____
4A39	T	_____
4A3A	O	_____
4A3B	N	_____
4A3C	space	_____
4A3D	A	_____
4A3E	N	_____
4A3F	D	_____
4A40	space	_____
4A41	C	_____
4A42	S	_____
4A43	A	_____
4A44	V	_____
4A45	E	_____
4A46	O	_____
4A47	!	_____

Figure 3.1 Worksheet for ASCII Codes

Since our message for the revision is longer than before, we'll also have to change the value in location 4A08 to 18 (there are 18 characters in our new message).

```
# M 4A08 0B 18        New value
          Old value
```

We're now ready to try the program. You should examine the memory to see that it has been entered correctly.

TO RUN THE PROGRAM

1. After typng X to clear tlie MEMORY mode, type: B4C00 tlien J4C00. This will clear tlie screen.
2. Put the recorder in the RECORD mode witli the cassette index at a new setting to record the program *and* the data.
3. Type: J4A00
   a. The program is run with your message displayed.

```
DEMO OF CTON AND CSAVEO!
```

   b. Press the @ key when you have finished viewing the screen.
   c. The recorder will start, and the program will be saved on tape.
   d. When the # sign appears, it is finished.

You may want to remove this added section when making future runs to avoid re-recording it every time the program is used. To do this, type: M4A17 and then CD 91 40 into locations 4A17, 4A18, and 4A19.

If you want to get real fancy, automatically load and run the program in the following manner:

1. Reset the recorder to the index setting where you started the recording made previously.
2. Access tlie MEMORY mode and type in:

```
# M 4A50 XX CD      Calls CTON to turn on
4A51 YY E9          recorder
4A52 XX 0F
```

4A53 YY CD	Calls CLOADO to
4A54 XX FA	send data from
4A55 YY 0E	recorder to computer
4A56 XX C3	
4A57 YY 00	After loading, jumps
4A58 XX 4A	to start of program
	and executes it

3. Type: X to get out of MEMORY mode.
4. Set the PLAY button on the recorder.
5. Then type: J 4A50 and the instructions entered in step 2 above will be executed followed by the program at 4A00.

SUMMING UP

Volume and tone levels necessary for saving and loading via tape recorders may vary from one recorder to another. Follow the directions furnished with the Radio Shack reference materials, but you may have to experiment to find the best settings for your recorder. The method that you use with the recorder will depend upon the version of BASIC which you have in your TRS-80.

It is convenient, if not necessary, to use some permanent storage device (such as a cassette recorder) as you are creating long programs. It is also a great timesaver when you want to re-enter some previously used program. Therefore it is important that you learn the recording and loading techniques well.

It is also important that you document the files of programs that you save. This should include: file name, recorder index settings (start and end), volume and tone levels used, and whatever else that you feel will be helpful.

THE FOURTH PROBLEM
Gaming

After three chapters of machine language programming, it's time for some fun and games. The recreational aspect of computing is usually one of the first areas investigated by the beginning programmer. We use it here as a break from our more serious endeavors.

Problem: Have some fun while learning to use T-BUG

How to Reach a Solution: Design a computer game that one lonely programmer can play when tired of serious learning. We'll use a number guessing game. You can match wits with the computer by trying to guess the number it is hiding deep down in its innards.

Your guess will be input from the keyboard. Then the computer will compare the input with its hidden number. Appropriate messages will appear on the screen to aid you in your quest.

NUMBER

Our guessing game has been used in many variations for some time. The originator's name has long been lost, but some variation of the game appears in almost every book of computer games and programs. The logic for our game is very simple, as shown in Figure 4.1.

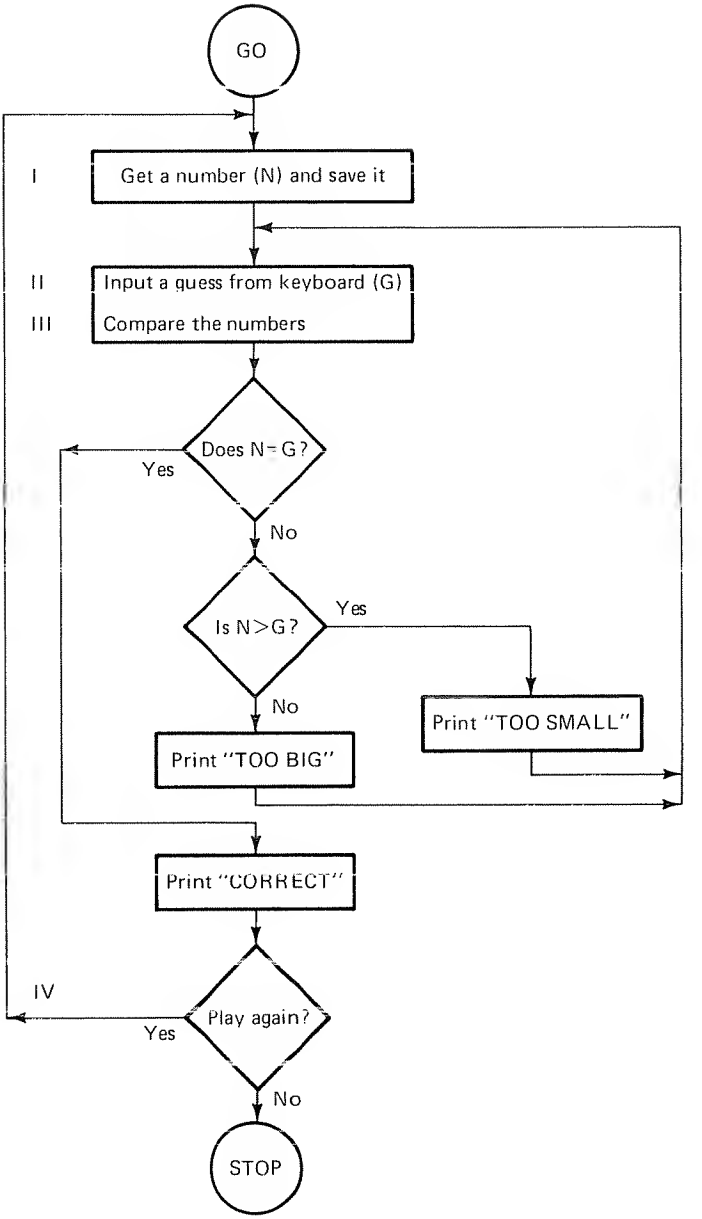


Figure 4.1 Flowchart for Number

HOW THE PROGRAM WORKS

We will discuss the program in four sections, as marked by the Roman numerals in Figure 4.1.

I. Computer Gets a Number
II. Input the Guess
III. Compare the Numbers
IV. Decide to Replay or Quit

I. Computer Gets a Number.

Where does the number that the computer hides come from? Rather than confuse the issue with a complicated random number generator, we have the compuer use a number from register R, the memory refresh counter. This register is not used by the programmer normally. It is a device for refreshing dynamic memories, if dynamic memories are used in the computer's hardware. Since the user is unaware of its contents, it is an ideal number generator for our game.

The current value of register R will require some programming manipulations to change it into two binary-coded digits so that the value may be interpreted as a two-digit decimal number. This value is then tucked away in memory location 4B20 for future use.

II. Input the Guess

When it is time for you to tell the computer your guess, it will print on the screen:

```
* NUMBER *
GUESS? _
```

It will then wait for you to type in your two-digit number. When you have entered the first digit, the computer calls the CHKIO subroutine which gets the character that you typed in (the tens digit of your guess) and echoes it on the screen. It changes the ASCII code of the digit by means of the AND and RLCA commands. The upper 4 bits of the ASCII code are stripped off, and the lower 4 bits are moved into the upper 4-bit position.

Example: 1. You type in the digit 5
2. ASCII code in accumulator = 35 (or 0011 0101 binary)
3. We then AND the accumulator with the value of 0F.
   a. 0011 0101 in accumulator
   b. 0000 1111 AND with 0F (hex)
   c. 0000 0101 Result left in accumulator. The upper 4 bits have been removed.
      (5)

Next, the RLCA command is used four times to move everything in the accumulator left 4 places (each use of RLCA moves each bit one place to the left. After 4 times, the lower 4 bits have been moved to the upper 4-bit positions as shown.

Example: 1. 0000 0101 in the accumulator from previous example
2. 0000 1010 after the 1st RLCA
3. 0001 0100 after the 2nd RLCA
4. 0010 1000 after the 3rd RLCA
5. 0101 0000 after the 4th RLCA
   (our 5 digit is now here)

Next, the computer calls the CHKIO subroutine again and gets your next input (the ones digit of your number) and prints it on the screen. It then takes the ASCII code for the digit you just typed in. This time it ANDs it with the hex number 0F, as before. This time, it does not rotate the bits. Instead, register C (our previous digit) is added to our new value. Now the accumulator holds what is called the binary-coded-decimal value of your original number.

Example: 1. You type in a second digit, 4
2. The ASCII code = 34
   a. (or 0011 0100 binary) in accumulator
   b. 0000 1111 AND with 0F
   c. 0000 0100 Result left in accumulator

3. 0101 0000 Value from register C to be added
4. 0101 0100 Result of addition (now in accumulator)
   5 4

The procedure discussed above is called *packing*. The process takes two ASCII numbers entered from the keyboard and combines them into one binary byte for use in the program. The "packed" number is then put back into register C for future use.

The computer then does a carriage return and line feed on the screen to get ready for the message which will tell you the result of your guess.

III. Compare the Numbers

This section of the program, although small, is probably the most important part of the game. The computer has chosen a number, N. You have input a guess, G. Three possible conditions may result:

1. N = G Your guess is correct
2. N>G Your guess is too small
3. N<G Your guess is too big

The computer must let you know which of the three conditions exists. It will print a message to inform you of the result. The decision as to which message to deliver is made from a comparison of the two values, N and G. The instruction which does the comparison is:

```
    CP C   where C is the hex value in the C register
   /   \
opcode  reg.
```

The operation which takes place when the instruction is executed is A − C. The result of the subtraction is discarded (the value in the accumulator and that in register C remain unchanged), but certain status flags (in the status register F) are set, or reset, according to the results of the comparison.

The status flags that concern us are:

1. The ZERO flag (Z) is set to 1 if the result of the comparison is zero (N = G). The ZERO flag is reset to 0 if the result is *not* zero (N>G or N<G).
2. The CARRY flag (C) is reset to 0 if no borrow occurs in

the subtraction (N>G). The CARRY flag is set to 1 if a borrow does occur (N<G).

Therefore, these three paths are provided in the program:

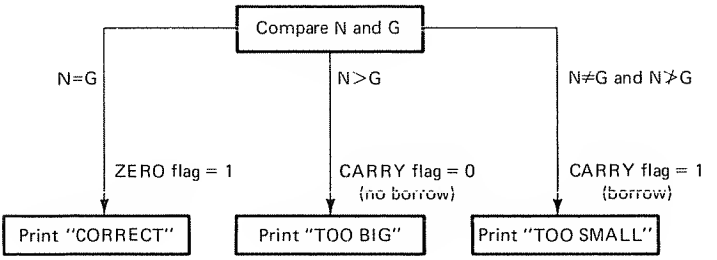


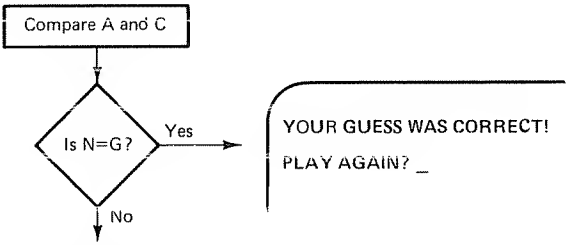
Figure 4.2 Paths of Number Comparison

Before the comparison is made, the computer has printed on the screen:

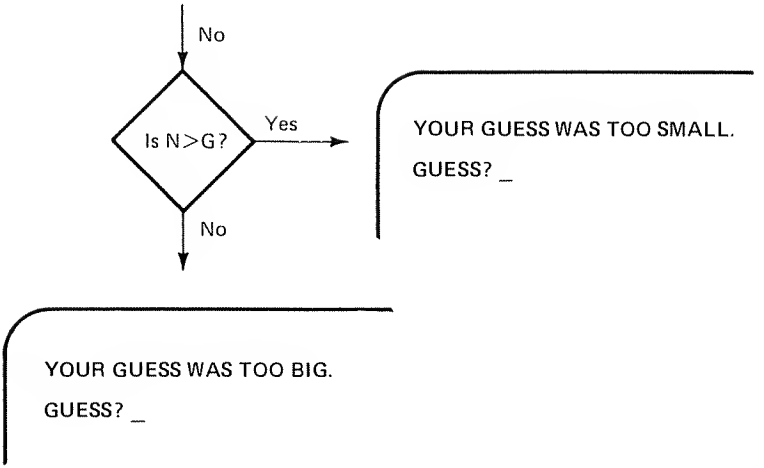
```
YOUR GUESS WAS
```

It then loads the computer's number, N, into the accumulator from memory (4B20) and compares it with your guess, G, which is in register C.

If the result is zero (both numbers are the same), the computer tells you that you were correct and asks you if you want to play again.



If the two numbers are not equal, the NO branch is taken. The computer then tests the CARRY flag. If the CARRY flag has been reset to 0, no borrow occurred (N − G is positive). Therefore N>G and the YES branch is taken. If the CARRY flag has been set to 1, a borrow did occur (N − G is negative). Therefore N<G and the NO branch is taken.



The computer, in either case, gives you the correct message and asks you for a new guess.

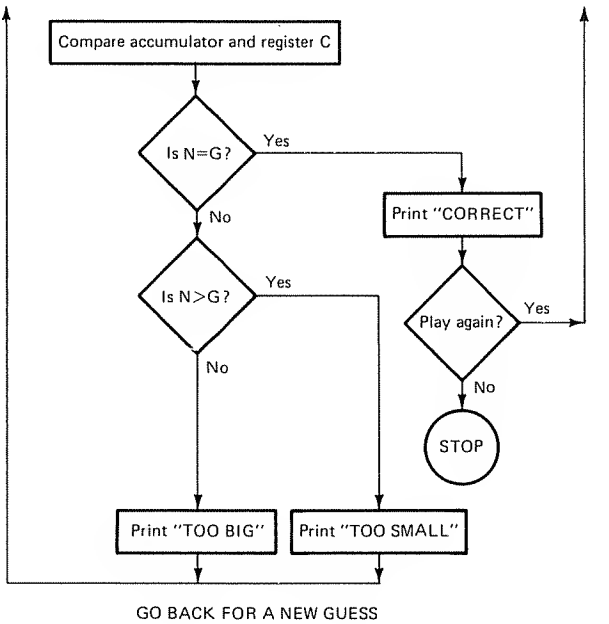


Figure 4.3 Flowchart for Compare Section

IV. Decide to Replay or Quit

There are two methods to exit from the program.

The first method is incorporated in that part of the program where you input your guess. Instead of typing in your two-digit number when the computer displays **GUESS?** __, merely type in the letter Q (for quit).

```
YOUR GUESS WAS TOO LOW.
GUESS? Q
```

The computer will then jump to that section of the program which displays:

```
PLAY AGAIN? __
```

Type: NO and control is given to the T-BUG monitor in the COMMAND mode.

The second method arises when you have made a correct guess. The computer displays:

```
YOUR GUESS WAS CORRECT!
PLAY AGAIN? __
```

Once again, type: NO and control returns to T-BUG.

The flowchart for the exit section of the program is shown in Figure 4.4.

Level II Program* — NUMBER

Level I users note changes in brackets given below.

I	II	III	IV	V	VI
LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
START	4A00	CD 96 4A	CALL	RANDOM	Get N
	4A03	21 C0 4A	LD	HL,4AC0	Print name of game
	4A06	06 0D	LD	B,0D	
	4A08	CD 8C 4A	CALL	PRINT	
INIT	4A0B	21 00 4B	LD	HL,4B00	Call for a guess and
	4A0E	06 09	LD	B,09	print it
	4A10	CD 8C 4A	CALL	PRINT	
	4A13	CD [2B 00]†	CALL	CHKIO	Clear accumulator
	4A16	AF	XOR	A	
DATA	4A17	CD [2B 00]†	CALL	CHKIO	Get input (G or Q)

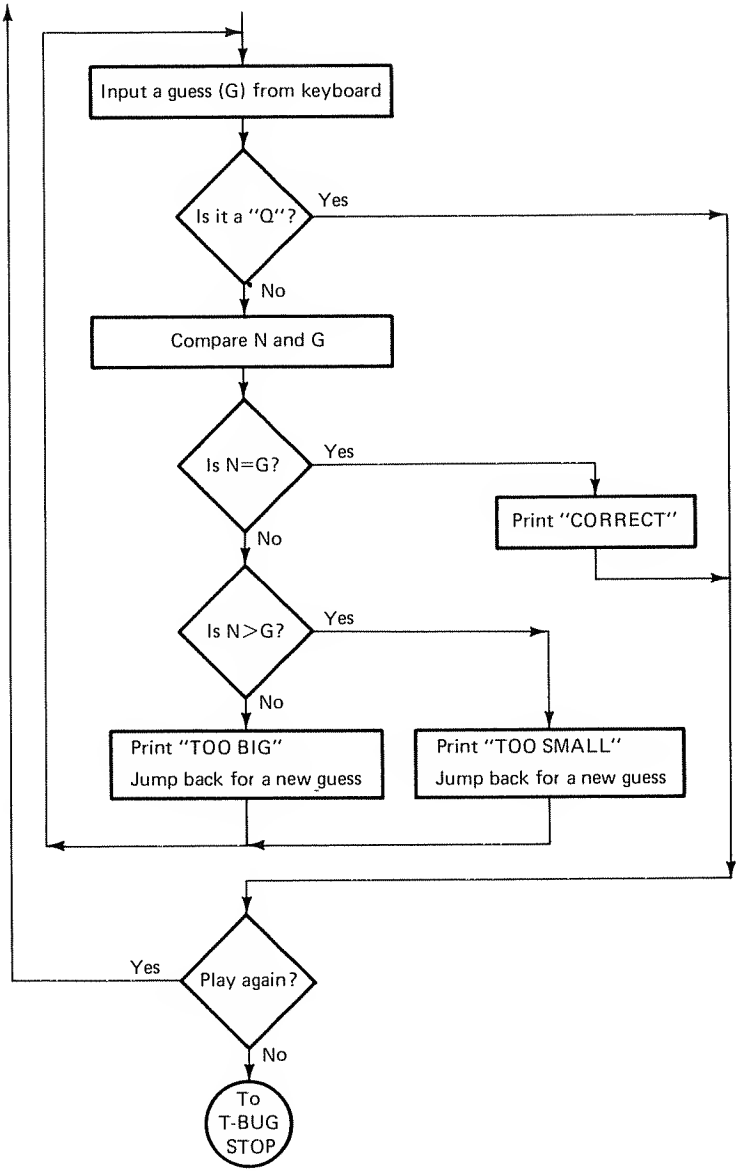


Figure 4.4 Flowchart for Exit Section

```
        4A1A  B7          OR    A
        4A1B  CA 17 4A    JP    Z,DATA
        4A1E  FE 51       CP    51
        4A20  CA 70 4A    JP    Z,QUIT
        4A23  [CD 33 00]² CALL  OUTC
        4A26  E6 0F       AND   0F        Strip upper 4 bits off
        4A28  07          RLCA            the ASCII tens digit
        4A29  07          RLCA            and rotate 4 places
        4A2A  07          RLCA            left
        4A2B  07          RLCA
        4A2C  4F          LD    C,A       Store it
DATA1   4A2D  CD [2B 00]¹ CALL  CHKIO     Now get ones digit
        4A30  B7          OR    A
        4A31  CA 2D 4A    JP    Z,DATA1
        4A34  [CD 33 00]² CALL  OUTC
        4A37  E6 0F       AND   0F
        4A39  81          ADD   A,C       Combine the digits
        4A3A  4F          LD    C,A       and save
        4A3B  3E 0D       LD    A,0D
        4A3D  [CD 33 00]² CALL  OUTC      Print "YOUR GUESS
        4A40  21 CD 4A    LD    HL,4ACD   WAS"
        4A43  06 11       LD    B,11
        4A45  CD 8C 4A    CALL  PRINT
        4A48  3A 20 4B    LD    A,(4B20)  Get number and guess
        4A4B  B9          CP    C         and compare them
        4A4C  CA 68 4A    JP    Z,OK      If N = G jump to OK
        4A4F  D2 5D 4A    JP    NC,SMALL  If N>G jump to
                                          SMALL
        4A52  21 E8 4A    LD    HL,4AE8   TOO BIG
        4A55  06 0B       LD    B,0B
        4A57  CD 8C 4A    CALL  PRINT
        4A5A  C3 0B 4A    JP    INIT      Try again
SMALL   4A5D  21 DE 4A    LD    HL,4ADE   TOO SMALL
        4A60  06 0A       LD    B,0A
        4A62  CD 8C 4A    CALL  PRINT
        4A65  C3 0B 4A    JP    INIT      Try again
OK      4A68  21 F3 4A    LD    HL,4AF3   CORRECT
        4A6B  06 0D       LD    B,0D
        4A6D  CD 8C 4A    CALL  PRINT
QUIT    4A70  21 09 4B    LD    HL,4B09   PLAY AGAIN?
        4A73  06 0E       LD    B,0E
        4A75  CD 8C 4A    CALL  PRINT
INPUT   4A78  CD [2B 00]¹ CALL  CHKID     INPUT yes or no
        4A7B  B7          OR    A
        4A7C  CA 78 4A    JP    Z,INPUT
        4A7F  FE 59       CP    59
        4A81  C4 [80 43]³ CALL  NZ,4380   To monitor if no
        4A84  3E 0D       LD    A,0D
        4A86  [CD 33 00]² CALL  OUTC
        4A89  C3 00 4A    JP    START
PRINT   4A8C  7E          LD    A,(HL)    MESSAGE
        4A8D  [CD 33 00]² CALL  OUTC
        4A90  05          DEC   B
        4A91  C8          RET   Z
        4A92  23          INC   HL
```

```
          4A93    C3 8C 4A   JP      PRINT
RANDOM    4A96    ED 5F      LD      A,R       Get a random number
          4A98    57         LD      D,A
          4A99    E6 0F      AND     0F        Lower digit
          4A9B    FE 0A      CP      0A
          4A9D    DA A3 4A   JP      C,NEXT
          4AA0    37         SCF               Set CARRY flag
          4AA1    D6 06      SUB     06        Subtract 6
NEXT      4AA3    5F         LD      E,A
          4AA4    7A         LD      A,D       Upper digit
          4AA5    E6 F0      AND     F0
          4AA7    FE A0      CP      A0
          4AA9    DA AF 4A   JP      C,DONE
          4AAC    37         SCF
          4AAD    D6 60      SUB     60
DONE      4AAF    83         ADD     A,E       Combine digits and
          4AB0    32 20 4B   LD      (4B20),A  store
          4AB3    C9         RET
```

* [2B00][1] Level I change to 40 0B
[CD 33 00][2] Level I change to D7 00 00
[80 43][3] Level I change to 91 40

ASCII Data for the Program

ADDRESS	DATA	MESSAGE
4AC0	2A 20 4E 55 4D 42 45 52 20 2A 0D 0A 0A	* NUMBER *
4ACD	0D 0A 59 4F 55 52 20 47 55 45 53 53 20 57 41 53 20	YOUR GUESS WAS
4ADE	54 4F 4F 20 4C 4F 57 2E 0D 0A	TOO LOW
4AE8	54 4F 4F 20 48 49 47 48 2E 0D 0A	TOO HIGH
4AF3	20 20 43 4F 52 52 45 43 54 21 0D 0A 0A	CORRECT!
4B00	0D 0A 47 55 45 53 53 3F 20	GUESS? _
4B09	0D 0A 50 4C 41 59 20 41 47 41 49 4E 3F 20	PLAY AGAIN? _

HOW TO PLAY NUMBER

In this game, you try to guess a decimal number. The range depends upon how much memory your machine has. The computer's number is taken from the memory refresh register, R. If you have a 4K machine, this number will range from 0 through 49. If you have a 16K machine, the number will range from 0 through 79.

1. Load the program and data via T-BUG as you have done before.
2. When step 1 has been completed, typing X will send you back to the monitor with the # sign displayed. Type: J 4A00. The computer will then ask for a guess by displaying:

```
* NUMBER *
GUESS? __
```

3. Enter your guess as a two-digit decimal number—do *not* press the ENTER key.
4. The computer will tell you that your guess was too high, too low, or correct.
5. a. If your guess was too high or too low, the computer will ask for another guess.

```
YOUR GUESS WAS TOO HIGH.
GUESS?__
```

   b. If your guess was correct, it will ask if you want to play again.

```
YOUR GUESS WAS CORRECT!
PLAY AGAIN? __
```

6. a. If the answer to 5b is **YES** (you typed YES and pressed ENTER), the computer will choose a new number and ask for a new guess.

```
* NUMBER *
GUESS? __
```

   b. If your answer to 5b is NO, you are returned to the T-BUG COMMAND mode.

```
#
```

7. At any time when asked for a new guess, you can end the game by typing Q. The computer will ask:

```
GUESS? Q
PLAY AGAIN? __
```

Type NO, and you will be back in T-BUG.

```
#
```

THE FIFTH PROBLEM
Drawing Your Own Graphics

T-BUG can be used effectively to create your own designs, display oversize characters, and other things on the video display. This can liven up your programs with dramatic visual effects.

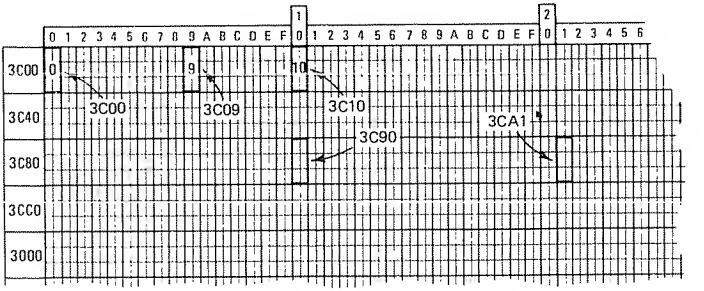
Problem: Create your own images on the video display.

How to Reach a Solution: The TRS-80 video display is divided into rectangles (128 horizontally and 64 vertically). If you have read *An Introduction to TRS-80 Graphics,* Don Inman, dilithium Press, you know how to create graphic displays using Level I BASIC. Each individual rectangle is turned ON or OFF by using the BASIC commands, SET and RESET. Machine language graphics can be performed more efficiently. Not only can you turn on individual rectangles, but you can also turn on blocks of 6 rectangles at once. Any combination of rectangles in the block of six can be lit (see Table IV, page 117 for the possibilities).

MEMORY MAP OF THE VIDEO DISPLAY

Memory locations 3C00 through 3FFF in the TRS-80 computer are used to display information on the video screen. Each block of six rectangles, 2 wide by 3 high, is assigned a unique location. On the Video Worksheet (page 61), the numbering system is shown starting with location 3C00 in the upper left corner. The memory locations are numbered successively from

left to right and top to bottom. Some typical locations are shown below.

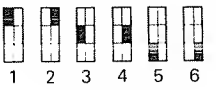


These memory locations are used to store ASCII values for the keyboard strokes, but they may also be used with special graphics codes. When this is done, use caution and stay within the boundaries of the locations assigned to the video display. Other nearby memory locations are used by the TRS-80 for other things.

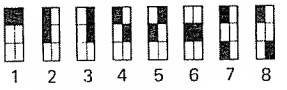
When the correct graphic code is stored in one of the memory locations, a unique pattern is lighted in that particular block of six rectangles. Your machine language program, input via T-BUG, can have complete control of the video display, and you can create your own designs.

THE GRAPHICS CODE

There are many combinations which can be used to make different patterns from six rectangles. Here are the ways:



Using one small rectangle—6 ways.



Two small rectangles—15 ways.

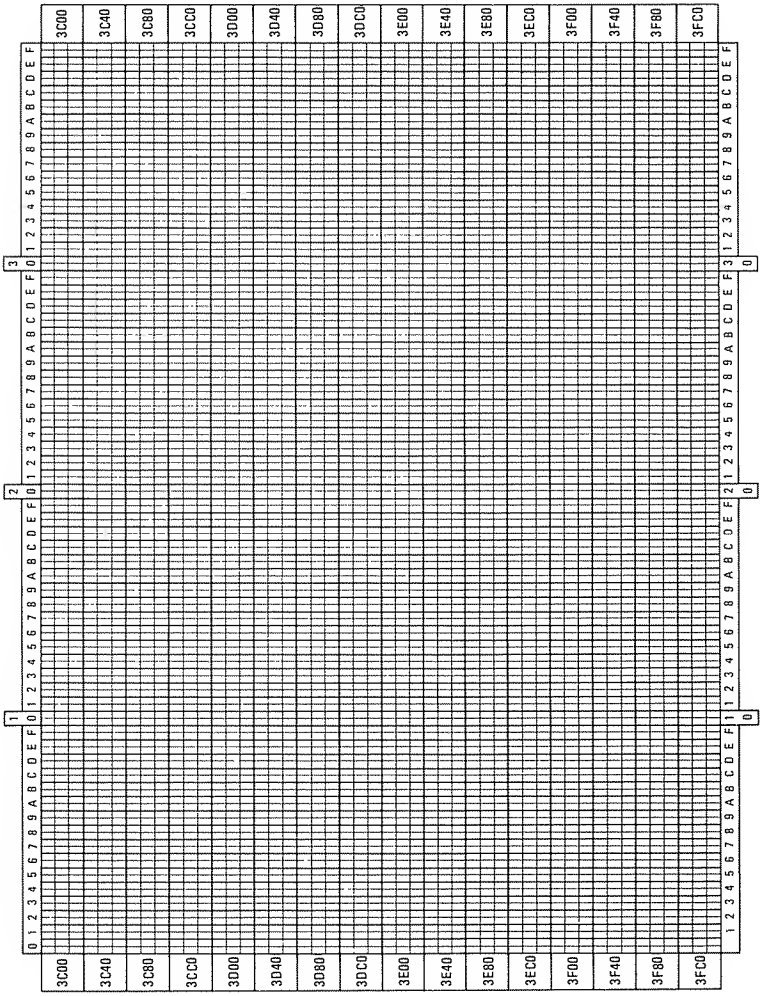
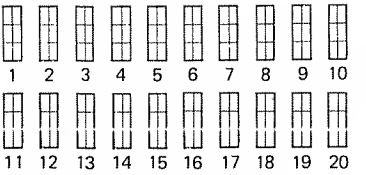
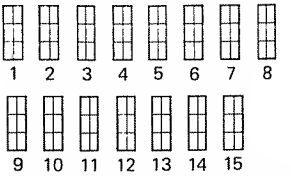


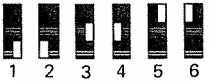
Figure 5.1 TRS-80 Video Display Worksheet



Using three small rectangles—20 ways. Fill them in.



Using four small rectangles—15 ways. Fill them in.



Five small rectangles—6 ways.

Six small rectangles—1 way.

If we add them all up, we find that 63 different arrangements are possible. Sixty-three is also the number of special graphic codes which the TRS-80 provides, one for each pattern. Therefore, we can display each of the 63 combinations of the six-block rectangles.

Each combination is designated by a unique hex number from 81 through BF. The complete table of patterns with their special hex codes are shown in Figure 5.2. The hex code is given directly below the pattern.

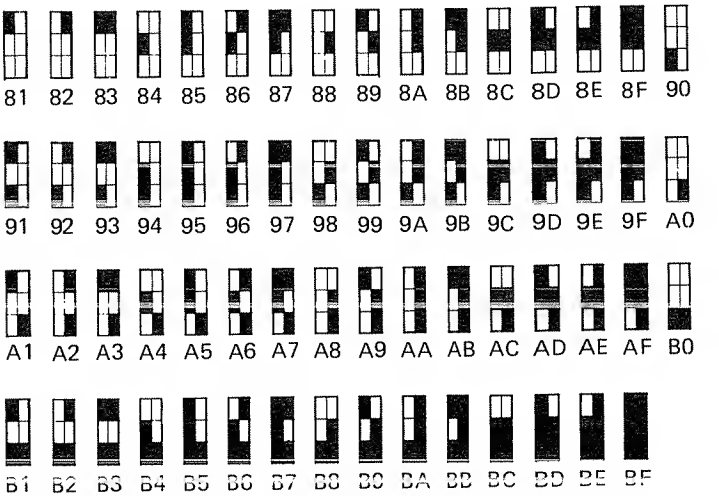


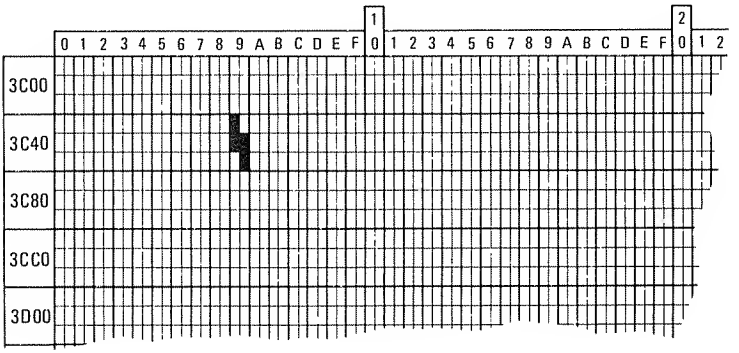
Figure 5.2 Graphic Codes

USING THE GRAPHIC CODES

Suppose that you wish to place the pattern of four rectangles whose code is AD near the left side of the second line of the video display. Choosing memory location 3C49, the following instructions provide *one* way to correctly light the screen.

3E AD	LD	A,AD	Load data into the accumulator
32 49 3C	LD	(Addr),A	Load accumulator's contents into memory (3C49)

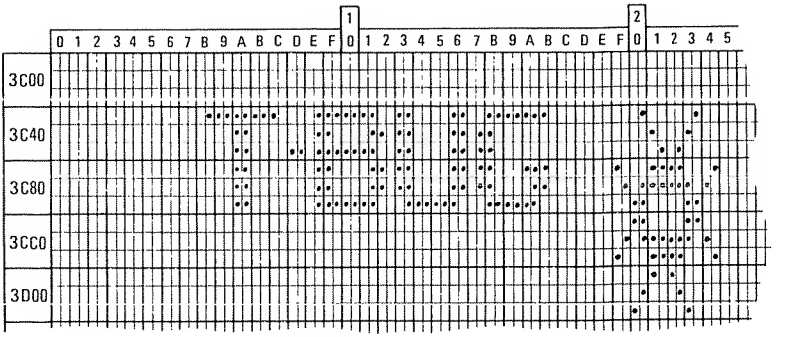
The execution of these instructions would place the pattern on the display in the position shown below.



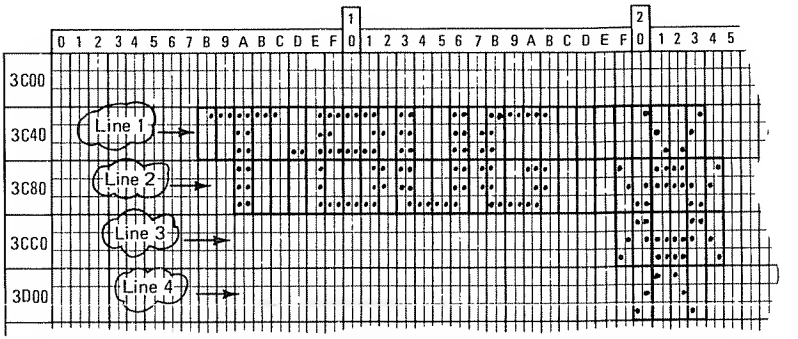
LAYING OUT A DESIGN

Using a TRS-80 Video Display Worksheet, as shown on page 00, a design can be made marking the individual rectangles that you want to light. These can then be sectioned into convenient 6-block units and be coded.

In the following example, I first put dots in each rectangle that I wanted to light.



I then sectioned off the 6-block large rectangles to correspond to their memory locations.



Using the graphic codes from Figure 5.2, I then made a table of codes needed for the memory locations used. Notice that there are some blanks in the code columns below. In those locations we want no colored small rectangles. We use the ASCII code for "space" (20) to place a blank in those locations. A printed ASCII character occupies a block of six small rectangles just as the graphic characters do.

Table of Codes and Display Memory Locations

Line 1		Line 2		Line 3	
Code	Memory	Code	Memory	Code	Memory
82	3C48	BF	3C8A	9B	3CDF
83	3C49		3C8B	8B	3CE0
BF	3C4A		3C8C	BC	3CE1
83	3C4B		3C8D	BC	3CE2
81	3C4C	AA	3C8E	87	3CE3
B0	3C4D	B5	3C8F	A4	3CE4
AA	3C4E	B0	3C90		
B7	3C4F	BA	3C91		
B3	3C50	85	3C92		
BB	3C51	AF	3C93		
84	3C52	B0	3C94		
BF	3C53	B0	3C95		
	3C54	9F	3C96		
	3C55	8A	3C97		
BF	3C56	B5	3C98		
A8	3C57	B0	3C99		
97	3C58	BB	3C9A		
83	3C59	85	3C9B		
83	3C5A		3C9C		
81	3C5B		3C9D		
	3C5C		3C9E		
	3C5D	89	3C9F		
	3C5E	B8	3CA0		
	3C5F	8F	3CA1		
82	3C60	8F	3CA2		
A4	3C61	B4	3CA3		
A0	3C62	86	3CA4		
86	3C63				

Line 4	
Code	Memory
98	3D20
81	3D21
89	3D22
90	3D23

WRITING THE PROGRAM

One way to provide data for a program is to store the data in a data table and use a register pair to point to the correct data as needed. We will do this. We will also use another register pair to access the correct memory location on the video display. Each line of 6-block rectangles will be loaded by a separate loop in our program. The program will be broken up to correspond to the four lines that we must enter.

The DE register pair is used to point to the desired data, and the HL register pair is used to tell where the data should be stored in the display memory. The accumulator will be used as a transfer location. The data is placed in the accumulator from the memory table and then transferred to the correct display location. The program begins at memory location 4A00, as usual. The data is stored in memory beginning at location 4B00.

The E register is tested in each loop to see whether all the data for that line has been entered or not. If it has, control is passed to the next loop.

The Level I program is shown first, and the Level II program follows. They are very similar. A new method to clear the screen is shown at the top of the Level II version.

Level I Graphics Program – 5A

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
START	4A00	21 48 3C	LD	HL,3C48	Load pointers
	4A03	11 00 4B	LD	DE,4B00	
LOOP1	4A06	1A	LD	A,(DE)	Load data
	4A07	77	LD	(HL),A	Display it
	4A08	23	INC	HL	Move pointers
	4A09	13	INC	DE	
	4A0A	7B	LD	A,E	Check for end of
	4A0B	FE 1C	CP	A,1C	data
	4A0D	C2 06 4A	JP	NZ,LOOP1	Go back if not done
	4A10	21 8A 3C	LD	HL,3C8A	Set pointer for line 2
LOOP2	4A13	1A	LD	A,(DE)	Load data
	4A14	77	LD	(HL),A	Display it
	4A15	23	INC	HL	
	4A16	13	INC	DE	
	4A17	7B	LD	A,E	Check for end of
	4A18	FE 37	CP	A,37	data
	4A1A	C2 13 4A	JP	NZ,LOOP2	Back if not done
	4A1D	21 DF 3C	LD	HL,3CDF	Pointer to line 3
LOOP3	4A20	1A	LD	A,(DE)	Load data
	4A21	77	LD	(HL),A	Display it
	4A22	23	INC	HL	
	4A23	13	INC	DE	
	4A24	7B	LD	A,E	Check for end of
	4A25	FE 3D	CP	A,3D	data
	4A27	C2 20 4A	JP	NZ,LOOP3	Back if not done
	4A2A	21 20 3D	LD	HL,3D20	Pointer to line 4
LOOP4	4A2D	1A	LD	A,(DE)	Load data
	4A2E	77	LD	(HL),A	Display it
	4A2F	23	INC	HL	

	4A30	13	INC	DE	
	4A31	7B	LD	A,E	Check for end of
	4A32	FE 41	CP	A,41	data
	4A34	C2 2D 4A	JP	NZ,LOOP4	Back if not done
ENDO	4A37	CD 40 0B	CALL	CHKIO	Leave display until
	4A3A	FE 40	CP	A,40	@ key is struck
	4A3C	C2 37 4A	JP	NZ,ENDO	
	4A3F	CD 91 40	CALL	MON	

Level II Graphics Program—5A

I	II	III	IV	V	VI
LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
START	4A00	21 00 3C	LD	HL,3C00	Clear screen by
LOOP0	4A03	3E 20	LD	A,20	filling it with
	4A05	77	LD	(HL),A	spaces until
	4A06	23	INC	HL	HL = 4000
	4A07	7C	LD	A,H	
	4A08	FE 40	CP	A,40	
	4A0A	C2 03 4A	JP	NZ,4A03	
	4A0D	21 48 3C	LD	HL,3C48	Display line 1
	4A10	11 00 4B	LD	DE,4B00	
LOOP1	4A13	1A	LD	A,(DE)	
	4A14	77	LD	(HL),A	
	4A15	23	INC	HL	
	4A16	13	INC	DE	
	4A17	7B	LD	A,E	
	4A18	FE 1C	CP	A,1C	
	4A1A	C2 13 4A	JP	NZ,LOOP1	
	4A1D	21 8A 3C	LD	HL,3C8A	Display line 2
LOOP2	4A20	1A	LD	A,(DE)	
	4A21	77	LD	(HL),A	
	4A22	23	INC	HL	
	4A23	13	INC	DE	
	4A24	7B	LD	A,E	
	4A25	FE 37	CP	A,37	
	4A27	C2 20 4A	JP	NZ,LOOP2	
	4A2A	21 DF 3C	LD	HL,3CDF	Display line 3
LOOP3	4A2D	1A	LD	A,(DE)	
	4A2E	77	LD	(HL),A	
	4A2F	23	INC	HL	
	4A30	13	INC	DE	
	4A31	7B	LD	A,E	
	4A32	FE 3D	CP	Z,3D	
	4A34	C2 2D 4A	JP	NZ,LOOP3	
	4A37	21 20 3D	LD	HL,3D20	Display line 4
LOOP4	4A3A	1A	LD	A,(DE)	
	4A3B	77	LD	(HL),A	
	4A3C	23	INC	HL	
	4A3D	13	INC	DE	
	4A3E	7B	LD	A,E	

```
        4A3F  FE 41    CP      A,41
        4A41  C2 3A 4A JP      NZ,LOOP4
ENDO    4A44  CD 2B 00 CALL    CHKIO    Wait for an @
        4A47  FE 40    CP      A,40
        4A49  C2 44 4A JP      NZ,ENDO
        4A4C  CD 80 43 CALL    MON
```

LOADING THE PROGRAM AND DATA

Now we're ready. Load T-BUG from cassette.
Type: M 4A00
Type: the program
Type: X (to exit memory mode)
Type: M 4B00
Type: the data which follows:

Line 1		Line 2		Line 3	
Address	Data	Address	Data	Address	Data
4B00	82	4B1C	BF	4B37	9B
4B01	83	4B1D	20	4B38	8B
4B02	BF	4B1E	20	4B39	BC
4B03	83	4B1F	20	4B3A	BC
4B04	81	4B20	AA	4B3B	87
4B05	B0	4B21	B5	4B3C	A4
4B06	AA	4B22	B0		
4B07	B7	4B23	BA		
4B08	B3	4B24	85		
4B09	BB	4B25	AF		
4B0A	84	4B26	B0	Line 4	
4B0B	BF	4B27	B0	Address	Data
4B0C	20	4B28	9F	4B3D	98
4B0D	20	4B29	8A	4B3E	81
4B0E	BF	4B2A	B5	4B3F	89
4B0F	A8	4B2B	B0	4B40	90
4B10	97	4B2C	BB		
4B11	83	4B2D	85		
4B12	83	4B2E	20		
4B13	81	4B2F	20		
4B14	20	4B30	20		
4B15	20	4B31	89		
4B16	20	4B32	B8		
4B17	20	4B33	8F		

4B18	82	4B34	8F
4B19	A4	4B35	B4
4B1A	A0	4B36	86
4B1B	86		

Type: X (to exit memory mode)
Type: J 4A00 and there is the display, quick as a wink!

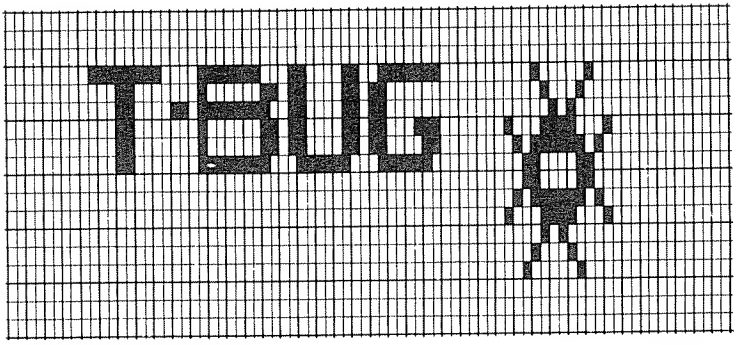


Figure 5.3 As Seen on the Display

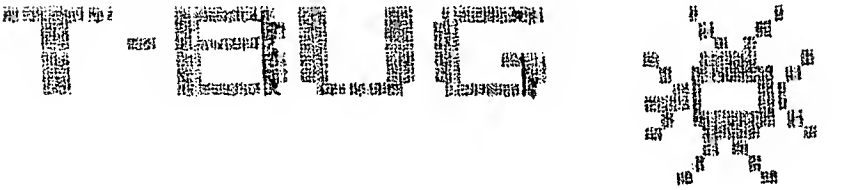


Figure 5.4 As Seen on the TRS-80 Screen Printer

A LARGER DISPLAY

In our second graphics program (5B), we will display the three figures shown in Figure 5.5. Each figure will be drawn separately. Below are the graphics codes for figures 5.5A and 5.5B. A table for you to complete for figure 5.5C is in Figure 5.6.

Codes for Figure 5.5A			Codes for Figure 5.5B				
Code	Memory		Code	Memory	Code	Memory	
82	3C48		BF	3C5B	B3	3D21	
AF	3C49	all			B3	3D22	
87	3C4A	BFs			AA	3D23	
			BF	3C64	BF	3D24	
B8	3C87		95	3C65	95	3D25	
BD	3C88						
AA	3C89		BF	3C9B	BF	3D5B	
A8	3C8A		BF	3C9C	BF	3D5C	
BD	3C8B		83	3C9D	AA	3D5D	
90	3C8C		B3	3C9E	A7	3D5E	
			B3	3C9F	BE	3D5F	
AF	3CC7		B3	3CA0	AA	3D60	
B2	3CC8		B3	3CA1	B6	3D61	
BA	3CC9		93	3CA2	AF	3D62	
B2	3CCA		AB	3CA3	AA	3D63	
BA	3CCB		BF	3CA4	BF	3D64	
85	3CCC		95	3CA5	95	3D65	
8B	3D08		BF	3CDB	BF	3D9B	
BF	3D09		BF	3CDC	BF	3D9C	
9F	3D0A		AA	3CDD	B0	3D9D	
81	3D0B		B6	3CDE	B3	3D9E	
			AF	3CDF	B3	3D9F	
AA	3D49		AA	3CE0	B3	3DA0	
			A7	3CE1	B3	3DA1	
A0	3D88		BE	3CE2	B1	3DA2	
BE	3D89		AA	3CE3	BA	3DA3	
B4	3D8A		BF	3CE4	BF	3DA4	
			95	3CE5	95	3DA5	
B8	3DC7						
BF	3DC8		BF	3D1B	BF	3DDB	all BFs
BF	3DC9		BF	3D1C			
BF	3DCA		A2	3D1D			
BD	3DCB		B3	3D1E	BF	3DE4	
90	3DCC		99	3D1F	95	3DE5	
			9D	3D20			

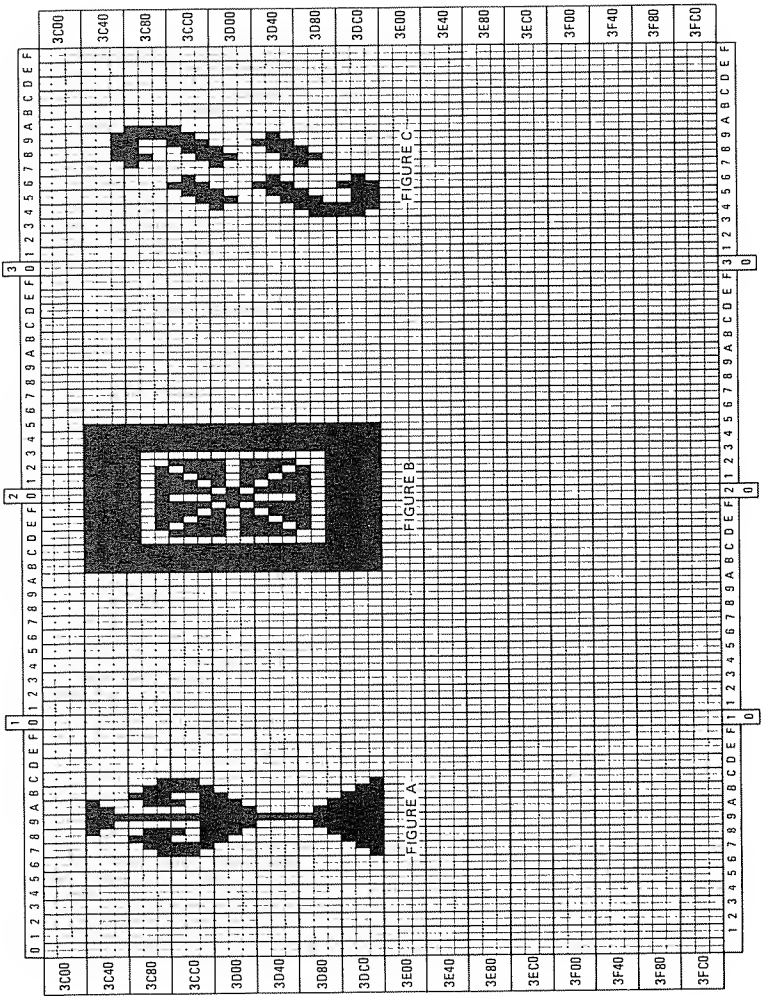


Figure 5.5 Video Display

CODE	MEMORY	CODE	MEMORY
B0	3C78		
	3C79		
	3C7A		
	3CB7		

Figure 5.6 Graphic Codes for Figure 5.5C

PROGRAM 5B

You probably noticed that there was much repetition in program 5A. Each loop, which created one line of the display, contained instructions which were also used in other loops. Program 5B shows a much more efficient way to handle these repetitions. The repeated instructions are placed in a subroutine which is called when needed. The values for the compare instruction, which change for each subroutine call, are contained in a table which is "pointed to" by the HL register pair.

This time we use three register pairs:

1. BC points to the memory location of the video screen.
2. DE points to the graphics code table.
3. HL points to the compare value table.

The program is made up largely of just two instructions:

1. LOAD the video memory location.
2. CALL the DISPLAY subroutine.

Let's start with the DISPLAY subroutine. It is similar to the instructions entered in each loop of program 5A, but it is entered in the computer only once this time. It will be used many times but it need only be entered once.

DISPLAY

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
LOOP1	4B00	1A	LD	A,(DE)	Load graphic code
	4B01	02	LD	(BC),A	Output to display
	4B02	03	INC	BC	
	4B03	13	INC	DE	
	4B04	7B	LD	A,E	See if more
	4B05	BE	CP	(HL)	
	4B06	C2 00 4B	JP	NZ,LOOP1	If more, go back
	4B09	23	INC	HC	Point to next data block
	4B0A	C9	RET		Return to main program

We will also use the clear screen technique of the Level II program 5A. This will be the entry point for running the program (4C00). Notice that it is not our first address used.

CLEAR SCREEN

I LABEL	II ADDRESS	III OPCODE	IV MNEMONIC	V OPERAND	VI REMARKS
	4C00	01 00 3C	LD	BC,3C00	Start at upper left
LOOP2	4C03	3E 20	LD	A,20	Put in a space
	4C05	02	LD	(BC),A	Put on screen
	4C06	03	INC	BC	
	4C07	78	LD	A,B	
	4C08	FE 40	CP	A,40	Look for end of CRT memory
	4C0A	C2 03 4C	JP	NZ,LOOP2	Go back if not there
	4C0D	C3 00 4A	JP	4A00	Go to main program

When we run the program, we start at 4C00, the clear screen portion. At the end of this section the program control is passed by the line at 4C0D to the main program.

The rest of the program is rather dull and monotonous. It is listed on the next two pages.

Main Program 5B (For Level I and Level II)

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4A00	01 48 3C	LD	BC,3C48	Display location
4A03	11 20 4B	LD	DE,4B20	Character location
4A06	21 0B 4B	LD	HL,4B0B	Compare value location
4A09	CD 00 4B	CALL	DISPLA	Display the character
4A0C	01 87 3C	LD	BC,3C87	Point to next character
4A0F	CD 00 4B	CALL	DISPLA	Display the character
4A12	01 C7 3C	LD	BC,3CC7	Point to next character
4A15	CD 00 4B	CALL	DISPLA	Display the character
4A18	01 08 3D	LD	BC,3D08	Point to next character
4A1B	CD 00 4B	CALL	DISPLA	Display the character
4A1E	01 49 3D	LD	BC,3D49	Point to next character
4A21	CD 00 4B	CALL	DISPLA	Display the character
4A24	01 88 3D	LD	BC,3D88	Point to next character
4A27	CD 00 4B	CALL	DISPLA	Display the character
4A2A	01 C7 3D	LD	BC,3DC7	Point to next character
4A2D	CD 00 4B	CALL	DISPLA	Display the character
4A30	01 5B 3C	LD	BC,3C5B	Point to next character
4A33	CD 00 4B	CALL	DISPLA	Display the character
4A36	01 9B 3C	LD	BC,3C9B	Point to next character
4A39	CD 00 4B	CALL	DISPLA	Display the character
4A3C	01 DB 3C	LD	BC,3CDB	Point to next character
4A3F	CD 00 4B	CALL	DISPLA	Display the character
4A42	01 1B 3D	LD	BC,3D1B	Point to next character
4A45	CD 00 4B	CALL	DISPLA	Display the character
4A48	01 5B 3D	LD	BC,3D5B	Point to next character
4A4B	CD 00 4B	CALL	DISPLA	Display the character
4A4E	01 9B 3D	LD	BC,3D9B	Point to next character
4A51	CD 00 4B	CALL	DISPLA	Display the character
4A54	01 DB 3D	LD	BC,3DDB	Point to next character
4A57	CD 00 4B	CALL	DISPLA	Display the character
4A5A	01 78 3C	LD	BC,3C78	Point to next character
4A5D	CD 00 4B	CALL	DISPLA	Display the character
4A60	01 B7 3C	LD	BC,3CB7	Point to next character
4A63	CD 00 4B	CALL	DISPLA	Display the character
4A66	01 F5 3C	LD	BC,3CF5	Point to next character
4A69	CD 00 4B	CALL	DISPLA	Display the character
4A6C	01 34 3D	LD	BC,3D34	Point to next character

4A6F	CD 00 4B	CALL	DISPLA	Display the character
4A72	01 75 3D	LD	BC,3D75	Point to next character
4A75	CD 00 4B	CALL	DISPLA	Display the character
4A78	01 B4 3D	LD	BC,3DB4	Point to next character
4A7B	CD 00 4B	CALL	DISPLA	Display the character
4A7E	01 F4 3D	LD	BC,3DF4	Point to next character
4A81	CD 00 4B	CALL	DISPLA	Display last character

We finish with a loop which looks for the character "@" to end the program. Level I and Level II versions differ in this loop.

```
LOOP3   4A84   CD 40 0B  CALL   CHKIO
        4A87   FE 40     CP     A,40          Level I
        4A89   C2 84 4A  JP     NZ,LOOP3      Only!
        4A8C   CD 91 40  CALL   MON

LOOP3   4A84   CD 2B 00  CALL   CHKIO
        4A87   FE 40     CP     A,40          Level II
        4A89   C2 84 4A  JP     NZ,LOOP3      Only!
        4A8C   CD 80 43  CALL   MON
```

That completes the program. Note the differences in LOOP3 for Level I and Level II users. Use the appropriate loop for your machine. After a brief rest, you should be ready to enter the graphic code data shown in Figure 5.7 and the compare value data table shown in Figure 5.8. This will give you a chance to see if you selected the correct codes for Figure 5.5C. Those codes begin at memory location 4B8A.

Address	Code	Address	Code	Address	Code	Address	Code	CRT*
4B20	82	4B43	BF	4B67	BF	4B8A	B0	3C78
4B21	AF	4B44	BF	4B68	95	4B8B	B0	3C79
4B22	87	4B45	BF			4B8C	20	3C7A
		4B46	BF	4B69	BF			
4B23	B8	4B47	95	4B6A	BF	4B8D	82	3CB7
4B24	D			4B6B	AA	4B8E	87	3CB8
4B25	AA	4B48	BF	4B6C	A7	4B8F	AB	3CB9
4B26	A8	4B49	BF	4B6D	BE	4B90	95	3CBA
4B27	BD	4B4A	83	4B6E	AA			
4B28	90	4B4B	B3	4B6F	B6	4B91	B8	3CF5
		4B4C	B3	4B70	AF	4B92	9D	3CF6
4B29	AF	4B4D	B3	4B71	AA	4B93	20	3CF7
4B2A	B2	4B4E	B3	4B72	BF	4B94	B8	3CF8
4B2B	BA	4B4F	93	4B73	95	4B95	9F	3CF9
4B2C	B2	4B50	AB			4B96	81	3CFA
4B2D	BA	4B51	BF	4B74	BF			

4B2E	85	4B52	95	4B75	BF	4B97	82	3D34
				4B76	B0	4B98	87	3D35
4B2F	8B	4B53	BF	4B77	B3	4B99	20	3D36
4B30	BF	4B54	BF	4B78	B3	4B9A	82	3D37
4B31	9F	4B55	AA	4B79	B3	4B9B	87	3D38
4B32	81	4B56	B6	4B7A	B3			
		4B57	AF	4B7B	B1	4B9C	B8	3D75
4B33	AA	4B58	AA	4B7C	BA	4B9D	9D	3D76
		4B59	A7	4B7D	BF	4B9E	20	3D77
4B34	A0	4B5A	BE	4B7E	95	4B9F	B8	3D78
4B35	BE	4B5B	AA			4BA0	9D	3D79
4B36	B4	4B5C	BF	4B7F	BF			
		4B5D	95	4B80	BF	4BA1	BE	3DB4
4B37	B8			4B81	BF	4BA2	87	3DB5
4B38	BF	4B5E	BF	4B82	BF	4BA3	20	3DB6
4B39	BF	4B5F	BF	4B83	BF	4BA4	82	3DB7
4B3A	BF	4B60	A2	4B84	BF	4BA5	87	3DB8
4B3B	BD	4B61	B3	4B85	BF			
4B3C	90	4B62	99	4B86	BF	4BA6	AF	3DF4
		4B63	9D	4B87	BF	4BA7	BC	3DF5
4B3D	BF	4B64	B3	4B88	BF	4BA8	9D	3DF6
4B3E	BF	4B65	B3	4B89	95			
4B3F	BF	4B66	AA					
4B40	BF							
4B41	BF							
4B42	BF							

The codes in these columns are the answers to the table in Figure 5.6. The display locations are given in the last column.

Figure 5.7 Graphics Code Data for Program 5B

Memory	Content
4B0B	23
4B0C	29
4B0D	2F
4B0E	33
4B0F	34
4B10	37
4B11	3D
4B12	48
4B13	53
4B14	5E
4B15	69
4B16	74
4B17	7F
4B18	8A
4B19	8D
4B1A	91

4B1B	97
4B1C	9C
4B1D	A1
4B1E	A6
4B1F	A9

Figure 5.8 Data Table for HL Pointer

USING PROGRAM 5B

We're now ready to try out the program. Follow the steps below:

1. Load the main part of the program (4A00-4A8E) using T-BUG.
2. Load the DISPLAY subroutine (4B00-4B0A).
3. Load the data (4B0B-4BA8).
4. Load the CLEAR SCREEN program (4C00-4C0F). Notice that at the end a JUMP is made to the main part of the program.
5. Type: X (to exit the MEMORY mode).
6. When the # sign appears, type: J 4C00.

The screen will immediately be filled with our three figures as shown in Figure 5.9.

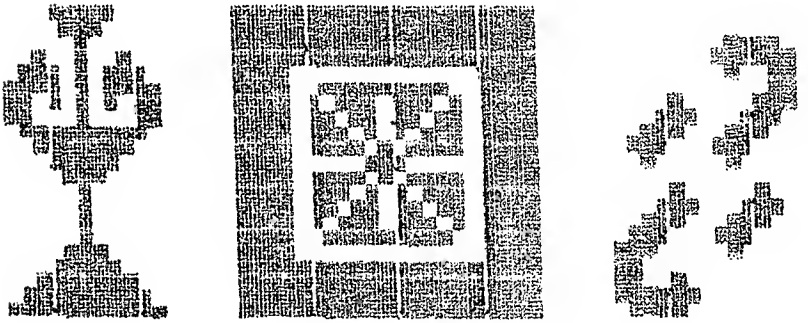


Figure 5.9 Display as Seen on the TRS-80 Screen Printer

THE SIXTH PROBLEM
Games Using Graphics

Recreational uses of computers inevitably lead to games, and the key to exciting games is the creative use of graphics.

Problem: Design some simple graphics to enhance a computer game.

How to Reach a Solution: We will use an old favorite, a version of the game of NIM, to demonstrate how graphics can be used in place of verbal clues and responses.

The screen displays an original pile of fifteen sticks which continually diminishes as the opposing players remove 1 to 3 sticks with each turn. The player who is forced to take the last stick loses.

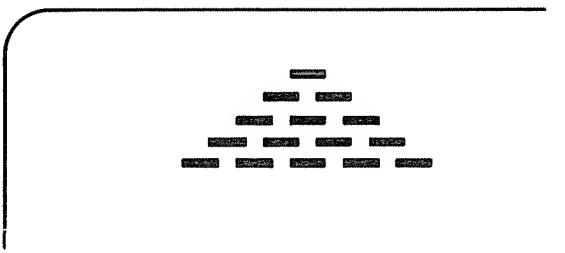
The program is rather long. Therefore we will explain each portion as it is encountered.

OUR GAME OF NIM

In this version of NIM, the game begins with a pile of fifteen sticks. It is played by one player who opposes the computer. Each player, in turn, removes from one to three sticks from the pile. The one who must take the last stick is the loser. The computer is always allowed to go first in our version.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
START	4A00	21 00 3C	LD	HL,3C00	Erase the display
LOOP0	4A03	3E 20	LD	A,20	
	4A05	77	LD	(HL),A	
	4A06	23	INC	HL	
	4A07	7C	LD	A,H	
	4A08	FE 40	CP	A,40	
	4A0A	C2 03 4A	JP	NZ,LOOP0	
	4A0D	26 3C	LD	H,3C	Draw the pile of
	4A0F	11 B0 4C	LD	DE,4CB0	sticks
	4A12	0E 0A	LD	C,0A	
	4A14	CD 1D 4C	CALL	LOOP1	
	4A17	26 3D	LD	H,3D	
	4A19	0E 05	LD	C,05	
	4A1B	CD 1D 4C	CALL	LOOP1	

Instructions at 4A00-4A0A clear the video display as before. The instructions 4A0D-4A1B draw a pile of 15 sticks for the start of the game. The subroutines named LOOP1 and LOOP1A actually construct the pile. LOOP1A is called from LOOP1.



LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A1E	21 9C 3D	LD	HL,3D9C	Label the pile
	4A21	11 E0 4C	LD	DE,4CE0	
	4A24	0E 08	LD	C,08	
	4A26	CD 14 4C	CALL	LOOP2	
	4A29	CD 33 4C	CALL	LOOP3	

Instructions 4A1E-4A26 print the message **THE PILE** under the pile of sticks. The instruction at 4A29 calls a subroutine

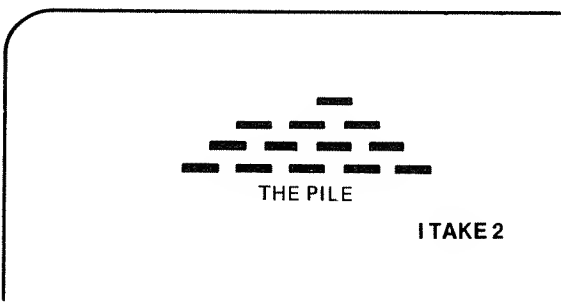
which performs a time delay so that the message can be read before the program proceeds.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A2C	CD F0 4C	CALL	LOOP4	Computer takes its
	4A2F	11 A8 4C	LD	DE,4CA8	turn
	4A32	1A	LD	A,(DE)	
	4A33	77	LD	(HL),A	

LOOP4 prints the message **I TAKE**, and the other three instructions add the value, 2, to the message so that it reads, **I TAKE 2.**

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A34	CD 33 4C	CALL	LOOP3	Erase two sticks
	4A37	06 5F	LD	B,5F	
	4A39	CD 65 4B	CALL	LOOP1A	

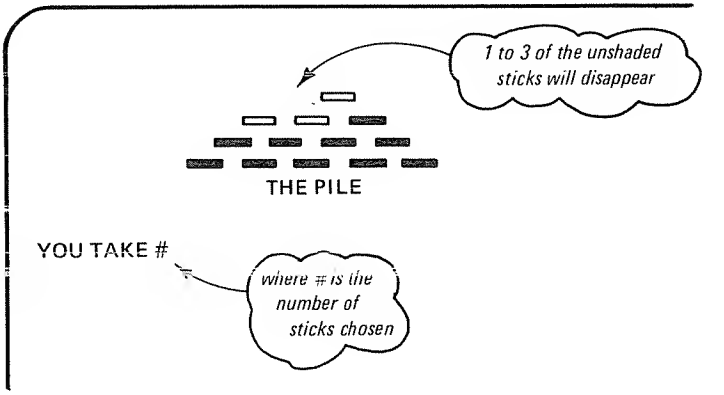
Loop 3 again provides a time delay, and the instructions at 4A37 and 4A39 erase two sticks leaving 13. The display shows:



LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A3C	FD 21 F0 4A	LD	IY,4AF0	Player's turn
	4A40	CD 00 4D	CALL	LOOP5	
	4A43	CD 10 4D	CALL	LOOP6	
	4A46	CD 50 4C	CALL	LOOP7	
	4A49	CD 33 4C	CALL	LOOP3	
	4A4C	CD 65 4B	CALL	LOOP7A	

The 4-byte instruction at 4A3C sets the index register, IY, to a memory location which holds the correct value to be used by register B for this stage of the program. LOOP5 prints **YOU TAKE ?** in the lower left part of the screen. LOOP6 erases **I TAKE 2** on the lower right. LOOP7 waits for the player to input his choice from the keyboard and prints it after **YOU TAKE ?** _. LOOP3 is the time delay again, and LOOP7A erases the correct number of sticks as chosen by the player.

You see on the screen:



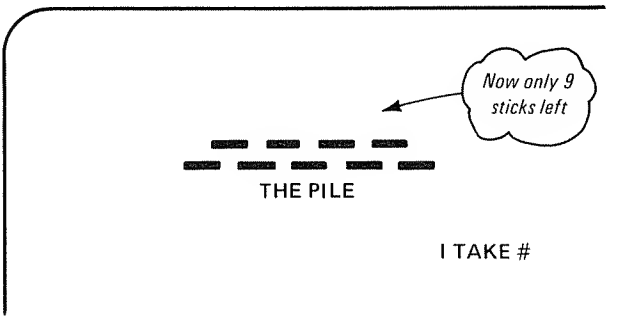
LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A4F	CD F0 4C	CALL	LOOP4	Computer's turn
	4A52	78	LD	A,B	
	4A53	FE 99	CP	A,99	
	4A55	CA 11 4B	JP	Z,ITAKE3 9	
	4A58	FE 9D	CP	A,9D	
	4A5A	CA 21 4B	JP	Z,ITAKE2	
	4A5D	C3 31 4B	JP	ITAKE1	

LOOP4 prints the message, **I TAKE** . The computer then examines the value in register B to see how many sticks were erased by the player's last choice. It will then take 1 of 3 branches depending upon that value. The computer always chooses a number that makes the sum of the player's input and its own choice equal to 4. The **ITAKE** branches print the appropriate number of the computer's choice and control returns to this next section.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
BACK	4A60	CD 20 4D	CALL	LOOP8	Erase sticks
	4A63	CD 33 4C	CALL	LOOP3	
	4A66	06 AF	LD	B,AF	
	4A68	CD 65 4B	CALL	LOOP7A	

LOOP8 erases **YOU TAKE #**, and LOOP3 gives a time delay. The other two instructions erase the appropriate number of sticks for the computer's choice.

The display now shows:



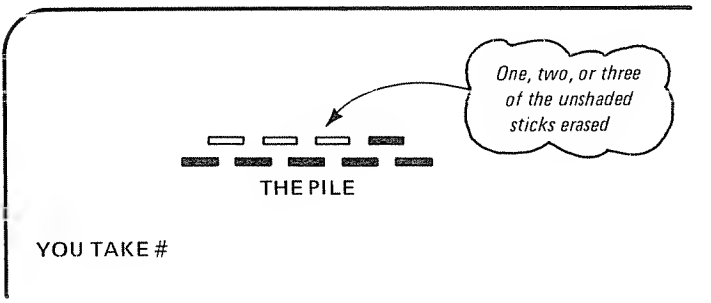
LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A6B	FD 21 F3 4A	LD	IV,4AF3	Change value for register B

The values used by register B must be changed so that the correct sticks will be wiped out following the player's next input.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A6F	CD 00 4D	CALL	LOOP5	Player's turn again
	4A72	CD 10 4D	CALL	LOOP6	
	4A75	CD 50 4C	CALL	LOOP7	
	4A78	CD 33 4C	CALL	LOOP3	
	4A7B	CD 65 4B	CALL	LOOP7A	

This is the same sequence of instructions used in the player's last turn (4A3C-4A48). One, two or three sticks are erased after

the player's choice is printed.



LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A7E	CD F0 4C	CALL	LOOP4	Computer's turn
	4A81	78	LD	A,B	
	4A82	FE DB	CP	DB	
	4A84	CA 41 4B	JP	Z,ITAKEC	
	4A87	FE DF	CP	E0 DF	
	4A89	CA 47 4B	JP	Z,ITAKEB	
	4A8C	C3 4D 4B	JP	ITAKEA	

Once again, LOOP4 prints **I TAKE** , and register B is compared to see what the player's choice was. The computer makes a choice, and control is returned to the next section.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
BACK1	4A8F	1A	LD	A,(DE)	Erase sticks
	4A90	77	LD	(HL),A	
	4A91	CD 20 4D	CALL	LOOP8	
	4A94	CD 33 4C	CALL	LOOP3	
	4A97	06 EF	LD	B,EF	
	4A99	CD 65 4B	CALL	LOOP7A	

LOOP8 wipes out **YOU TAKE #**, LOOP3 gives a time delay, and the other two instructions erase the appropriate number of sticks. Only the last row of five sticks is now left.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A9C	FD 21 F6 4A	LD	IY,4AF6	

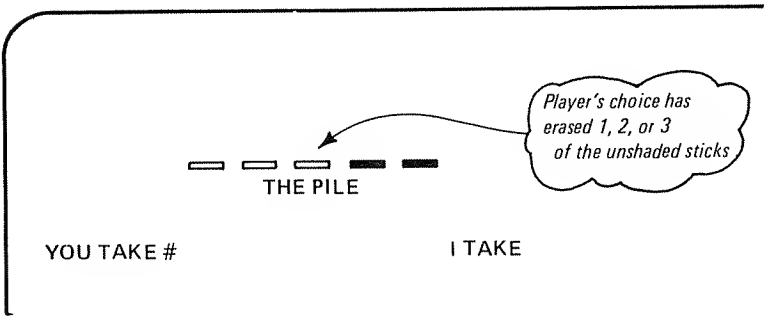
Once again the index register is changed to provide new values for register B for the player's last turn.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4AA0	CD 00 4D	CALL	LOOP5	Player's turn
	4AA3	CD 10 4D	CALL	LOOP6	
	4AA6	CD 50 4C	CALL	LOOP7	
	4AA9	CD 33 4C	CALL	LOOP3	
	4AAC	CD 6C 4B	CALL	LOOP7B	

LOOP5 prints **YOU TAKE ?**, and LOOP6 erases **I TAKE #**. LOOP7 waits for the input and prints it. LOOP3 provides a time delay, and LOOP7B erases the appropriate number of sticks.

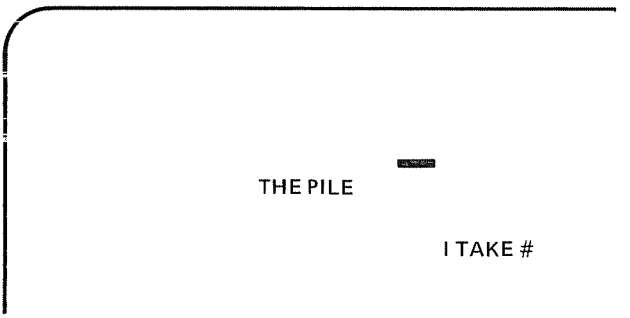
LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4AAF	CD F0 4C	CALL	LOOP4	Computer raps it up
	4AB2	78	LD	A,B	
	4AB3	FE 19	CP	19	
	4AB5	CA 53 4B	JP	Z,ITAKECC	
	4AB8	FE 1D	CP	1D	
	4ABA	CA 59 4B	JP	Z,ITAKEBB	
	4ABD	C3 5F 4B	JP	ITAKEAA	

LOOP4 prints **I TAKE** , and register B is examined to see what the player's choice was. The branch taken prints the computer's number and control is returned to the next section. Here is the present status of the display:



LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
BACK2	4AC0	1A	LD	A,(DE)	Finish him off
	4AC1	77	LD	(HL),A	
	4AC2	CD 20 4D	CALL	LOOP8	
	4AC5	CD 33 4C	CALL	LOOP3	
	4AC8	06 25	LD	B,25	
	4ACA	CD 6C 4B	CALL	LOOP7B	

The two instructions at 4AD2 and 4AD3 print the computer's number. LOOP8 wipes out **YOU TAKE #**, and LOOP3 gives a time delay. LOOP7B then erases all the sticks in the last row except one. The player has now clearly lost the game. This is how the screen looks:



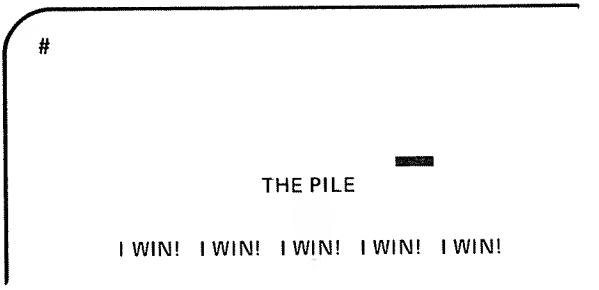
LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4ACD	CD 33 4C	CALL	LOOP3	Give a victory
	4AD0	06 07	LD	B,07	message
	4AD2	21 03 3F	LD	HL,3F03	
ROUND	4AD5	11 D0 4C	LD	DE,4CD0	
	4AD8	0E 07	LD	C,07	
	4ADA	CD 14 4C	CALL	LOOP2	
	4ADD	05	DEC	B	
	4ADE	C2 D5 4A	JP	NZ,ROUND	

This section prints the victory message on the screen.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
ENDALL	4AE1	CD [80 43]*	CALL	MON	End of program

* [91 40] for Level I

The control is passed back to the T-BUG monitor. The game is over and the final display shows:



Data for Register B

Address	Data
4AF0	99
4AF1	9D
4AF2	A1
4AF3	DB
4AF4	DF
4AF5	E3
4AF6	19
4AF7	1D
4AF8	21

This data provides stopping places for erasure of sticks when the player inputs his choices.

The program contains some jumps to the following sections which print the number selected by the player's input or to set the DE register pair to select the correct number.

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
ITAKE3	4B11	11 A9 4C	LD	DE,4CA9	Print a 3
	4B14	1A	LD	A,(DE)	
	4B15	77	LD	(HL),A	
	4B16	C3 60 4A	JP	BACK	
ITAKE2	4B21	11 A8 4C	LD	DE,4CA8	Print a 2
	4B24	1A	LD	A,(DE)	
	4B25	77	LD	(HL),A	
	4B26	C3 60 4A	JP	BACK	

ITAKE 1	4B31	11 A7 4C	LD	DE,4CA7	Print a 1
	4B34	1A	LD	A,(DE)	
	4B35	77	LD	(HL),A	
	4B36	C3 60 4A	JP	BACK	
ITAKEC	4B41	11 A9 4C	LD	DE,4CA9	Ready for 3
	4B44	C3 8F 4A	JP	BACK1	
ITAKEB	4B47	11 A8 4C	LD	DE,4CA8	Ready for 2
	4B4A	C3 8F 4A	JP	BACK1	
ITAKEA	4B4D	11 A7 4C	LD	DE,4CA7	Ready for 1
	4B50	C3 8F 4A	JP	BACK1	
ITAKECC	4B53	11 A9 4C	LD	DE,4CA9	Ready for 3
	4B56	C3 C0 4A	JP	BACK2	
ITAKEBB	4B59	11 A8 4C	LD	DE,4CA8	Ready for 2
	4B5C	C3 C0 4A	JP	BACK2	
ITAKEAA	4B5F	11 A7 4C	LD	DE,4CA7	Ready for 1
	4B62	C3 C0 4A	JP	BACK2	

That's the program except for the subroutines which follow in order of their appearance in the main program.

LOOP1

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4C1D	1A	LD	A,(DE)	Constructs the pile of
4C1E	6F	LD	L,A	15 sticks. Calls
4C1F	13	INC	DE	LOOP1A to draw each
4C20	CD 00 4C	CALL	LOOP1A	individual stick.
4C23	0D	DEC	C	
4C24	C2 1D 4C	JP	NZ,4C1D	
4C27	C9	RET		

LOOP1A

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4C00	06 03	LD	B,03	Draw a stick at the
4C02	3E 83	LD	A,83	location pointed to by
4C04	77	LD	(HL),A	the HL register pair.
4C05	23	INC	HL	
4C06	05	DEC	B	
4C07	C2 02 4C	JP	NZ.4C02	
4C0A	C9	RET		

LOOP2

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4C14	1A	LD	A,(DE)	Print the message
4C15	77	LD	(HL),A	pointed to by the DE
4C16	13	INC	DE	register pair on the
4C17	23	INC	HL	display at the location
4C18	0D	DEC	C	pointed to by HL.
4C19	C2 14 4C	JP	NZ,4C14	
4C1C	C9	RET		

LOOP3

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4C33	DD 21 FF FF	LD	IX,FFFF	A new 4-byte
4C37	DD 2B	DEC	IX	instruction. Index
4C39	DD 22 E8 4C	LD	(4CE8),(IX)	register (IX) used as a
4C3D	3A E9 4C	LD	A,(4CE9)	counter in the time
4C40	FE 00	CP	A,00	delay.
4C42	C2 37 4C	JP	NZ,4C37	
4C45	C9	RET		

LOOP4

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4CF0	21 29 3F	LD	HL,3F29	Loads HL and DE
4CF3	11 A0 4C	LD	DE,4CA0	register pairs to
4CF6	0E 07	LD	C,07	correct locations for
4CF8	CD 14 4C	CALL	LOOP2	printing a message.
4CFB	C9	RET		Register C provides correct character count.

LOOP5

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4D00	21 07 3F	LD	HL,3F07	Prints the words
4D03	11 C0 4C	LD	DE,4CC0	"YOU TAKE ?"
4D06	0F 09	LD	C,09	
4D08	CD 14 4C	CALL	LOOP2	
4D0B	C9	RET		

LOOP6

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4D10	21 29 3F	LD	HL,3F29	Erases the message
4D13	0E 0C	LD	C,0C	"I TAKE #" using HL
4D15	3E 20	LD	A,20	to point to the display
4D17	77	LD	(HL),A	location. C is a count
4D18	23	INC	HL	for the number of
4D19	0D	DEC	C	characters.
4D1A	C2 17 4D	JP	NZ,4D17	
4D1D	C9	RET		

LOOP7

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4C50	CD [2B 00]*	CALL	CHKIO	Waits for a keyboard
4C53	AF	XOR	A	input. When received,
4C54	CD [2B 00]*	CALL	CHKIO	compares it with
4C57	FE 31	CP	31	ASCII codes for 1, 2,
4C59	CA 69 4C	JP	Z,4C69	or 3, and takes the
4C5C	FE 32	CP	32	appropriate jump to
4C5E	CA 7B 4C	JP	Z,4C7B	print the value
4C61	FE 33	CP	33	selected.
4C63	CA 8C 4C	JP	Z,4C8C	
4C66	C3 54 4C	JP	4C54	
4C69	00 00 00	NOP		These NOPs do
4C6C	00 00 00	NOP		nothing. Some unused
4C6F	21 10 3F	LD	HL,3F10	instructions have
4C72	11 A7 4C	LD	DE,4CA7	been removed.
4C75	1A	LD	A,(DE)	Program comes here
4C76	77	LD	(HL),A	if input is a 1.
4C77	FD 46 00	LD	B,(IY + 0)	
4C7A	C9	RET		
4C7B	00 00	NOP		More NOPs that do
4C7D	00 00 00	NOP		nothing. Program
4C80	21 10 3F	LD	HL,3F10	comes here if input is
4C83	11 A8 4C	LD	DE,4CA8	a 2.
4C86	1A	LD	A,(DE)	
4C87	77	LD	(HL),A	
4C88	FD 46 01	LD	B,(IY + 1)	
4C8B	C9	RET		
4C8C	00 00	NOP		More NOPs. Program
4C8E	00 00 00	NOP		comes here if input is
4C91	21 10 3F	LD	HL,3F10	a 3.
4C94	11 A9 4C	LD	DE,4CA9	
4C97	1A	LD	A,(DE)	

4C98	77	LD	(HL),A
4C99	FD 46 02	LD	B,(IY + 2)
4C9C	C9	RET	

* [40 0B] for Level I

LOOP7A

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4B65	21 00 3C	LD	HL,3C00	Sets up area for
4B68	CD 73 4B	CALL	LOOP7C	erasure of sticks.
4B6B	C9	RET		

LOOP7B

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4B6C	21 00 3D	LD	HL,3D00	Sets up area for
4B6F	CD 73 4B	CALL	LOOP7C	erasure of sticks.
4B72	C9	RET		

LOOP7C

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4B73	3E 20	LD	A,20	Erases sticks by
4B75	77	LD	(HL),A	loading spaces into
4B76	23	INC	HL	appropriate display
4B77	7D	LD	A,L	memory.
4B78	B8	CP	B	
4B79	C2 73 4B	JP	NZ,4B73	
4B7C	C9	RET		

LOOP8

ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
4D20	21 07 3F	LD	HL,3F07	Sets up display
4D23	C3 13 4D	JP	4D13	address, then jumps into LOOP6.

DATA FOR THE MESSAGES

This final section provides the data for the messages which are printed by the program.

Address	Code	Character
4CA0	49	I
4CA1	20	space
4CA2	54	T
4CA3	41	A
4CA4	4B	K
4CA5	45	E
4AA6	20	space
4CA7	31	1
4CA8	32	2
4CA9	33	3
4CC0	59	Y
4CC1	4F	O
4CC2	55	U
4CC3	20	space
4CC4	54	T
4CC5	41	A
4CC6	4B	K
4CC7	45	E
4CC8	3F	?
4CD0	49	I
4CD1	20	space
4CD2	57	W
4CD3	49	I
4CD4	4E	N
4CD5	21	!
4CD6	20	space
4CE0	54	T
4CE1	48	H
4CE2	45	E
4CE3	20	space
4CE4	50	P
4CE5	49	I
4CE6	4C	L
4CE7	45	E

This version of NIM is not a game you will enjoy playing as there is no chance for success. The computer is in complete control at all times since it gets to make the first move. The game is presented here as a vehicle to show how graphics can enhance a game.

The program is long, and the possibilities for errors in entering it are many. You may want to go on to the next problem on debugging techniques before tackling such a long program. I made many mistakes and many revisions in writing the program. T-BUG was a great help.

There is much room for improvement in the program, and you may find that you wish to revise it. You can change the number of sticks. You could even desing an option as to who gets to make the move. Then you could beat the computer.

A good exercise is to try to add graphics to other computer games available in books, magazines, club newspapers, or wherever you find them.

THE SEVENTH PROBLEM
Debugging with T-BUG

What do you do when your program doesn't work in the way that you think it ought to? Was it your program logic, or did you make mistakes when you entered the program? T-BUG has some features to help you out.

Problem: A program has been written to solve a problem, but it fails to reach the correct solution.

How to Reach a Solution: If a program doesn't perform the job it was intended to do: either the user did not design the program correctly, or the user entered some faulty values. Program debugging is aided by some features of T-BUG. The BREAK command described on page 4 of the T-BUG users manual is one. The REGISTER command described on pages 4 and 5 of the same manual is another.

We will look at a faulty program and see how to use both of those commands in correcting program errors.

A BAD PROGRAM

Suppose that we have written a program to input a decimal digit (0-9) from the keyboard and print out the ASCII code for that digit.

Three errors will be incorporated in the program for debugging purposes. We won't tell you where they are, but their presence is rather obvious. The program is supposed to work like this:

1. You input a digit (0-9) from the keyboard.
2. Stored in memory is a message which the computer will print out:

3. The program then waits for you to enter one of two possible choices:
   a. Type: ENTER if you wish another entry;
   b. Type: @ to get back to T-BUG.

The program uses the HL register pair to point to the ASCII data of the message, stored in memory locations 4B01 through 4B17. The DE register pair points to the video display locations. Register B is used to temporarily store your digit while the accumulator is busy doing other tasks. For purposes of clarity, we'll make the Level I and Level II programs as nearly alike as possible.

We will use the BREAK and REGISTER commands to debug the program. Notice that a breakpoint can only be inserted in one place at a time. You then remove it and move it to the next desired location.

We have broken the program up into its five functional parts. The labels for these sections are LOOP0, LOOP1, LOOP2, LOOP3, and LOOP4. The remarks explain the functions of each section.

The program is written for Level II BASIC, but only four changes are needed for Level I. These changes are listed at the bottom of the program.

THE PROGRAM—#7

LABEL	ADDRESS	OPCODE	MNEMONIC	OPERAND	REMARKS
	4A00	21 00 3C	LD	HL,3C00	This section clears the
LOOP0	4A03	3E 20	LD	A,20	screen for the
	4A05	77	LD	(HL),A	message by filling it
	4A06	23	INC	HL	with blank spaces.
	4A07	7C	LD	A,H	
	4A08	FE 40	CP	A,40	
	4A0A	C2 03 4A	JP	NZ,LOOP0	
	4A0D	21 01 4B	LD	HL,4B01	This loop waits for you
	4A10	CD [2B 00]	CALL	CHKIO	to enter a decimal
	4A13	AF	XOR	A,A	digit, 0-9.

LOOP1	4A14	CD [2B 00]¹	CALL	CHKIO	
	4A17	FE 00	CP	A,00	
	4A19	CA 14 4A	JP	Z,LOOP1	
	4A1C	11 40 3C	LD	DE,3C40	This loop prints the
	4A1F	4F	LD	B,A	first part of message:
LOOP2	4A20	7E	LD	A,(HL)	"THE ASCII CODE
	4A21	12	LD	(DE),A	FOR ".
	4A22	13	INC	DE	
	4A23	23	INC	HL	
	4A24	7D	LD	A,L	
	4A25	FE 14	CP	A,14	
	4A27	C2 20 4A	JP	NZ,LOOP2	
	4A2A	78	LD	A,B	This prints your digit
	4A2B	12	LD	(DE),A	and the word "IS ". On
	4A2C	13	INC	DE	the screen now: "THE
LOOP3	4A2D	5E	LD	A,(HL)	ASCII CODE FOR X
	4A2E	12	LD	(DE),A	IS " (where X is your
	4A2F	13	INC	DE	digit).
	4A30	23	INC	HL	
	4A31	7D	LD	A,L	
	4A32	FE 17	CP	A,17	
	4A34	C2 2D 4A	JP	NZ,LOOP3	
	4A37	3E 33	LD	A,33	This adds a "3"
	4A39	12	LD	(DE),A	followed by your digit
	4A3A	13	INC	DE	and the final message
	4A3B	78	LD	A,B	"THE ASCII CODE
	4A3C	12	LD	(DE),A	FOR X IS 3X"
LOOP4	4A3D	CD [2B 00]¹	CALL	CHKIO	Make a choice: Type:
	4A40	FE 0D	CP	A,0D	ENTER to go back or
	4A42	CA 00 4A	JP	Z,4A00	@ to T-BUG.
	4A45	FE 40	CP	A,40	
	4A47	C2 3D 4A	JP	NZ,4A32	
	4A4A	CD [80 43]²	CALL	MON	

Level I users: Substitute [40 0B]¹; [91 40]²

DATA FOR THE BAD PROGRAM

Address	Code	Character
4B01	54	T
4B02	48	H
4B03	45	E
4B04	20	space
4B05	41	A
4B06	53	S
4B07	43	C
4B08	49	I
4B09	49	I
4B0A	20	space
4B0B	43	C
4B0C	4F	O

4B0D	44	D
4B0E	45	E
4B0F	20	space
4B10	46	F
4B11	4F	O
4B12	52	R
4B13	20	space
4B14	20	space
4B15	49	I
4B16	53	S
4B17	20	space

GETTING READY

Load the program, then the data, and finally type: X to exit the MEMORY mode.

DEBUGGING THE PROGRAM

First try to run the program by typing: J 4A00. The screen goes blank, waiting for your digit. Type in a 5. When I typed the 5 my screen showed:

```
                                  2
THE ASCIIUCODE FOR V32
```

The 2 used in this example may be any digit from 0-9.

Something is wrong! Looking back at the program, I see that:

1. LOOP0 which clears the screen worked perfectly.
2. LOOP1 waited for the digit to be entered and then went on as it was supposed to do.
3. LOOP2 should save our digit in register B and print **THE ASCII CODE FOR** .

Obviously, our program went astray in LOOP2. That may not be the only place, but that is where we should start looking.

Somehow, a **U** was inserted between the words **ASCII** and **CODE**. The **V32** and the extraneous **2** in the first line also look suspicious.

We will put a breakpoint in the program immediately following the instruction which moved our digit into register B. We

can then examine register B with the REGISTER command to see what was placed there.

FIRST TRY AT DEBUGGING

Type: @ to get back to the monitor. The # sign will appear. Now, type: B 4A20.

Your screen will look something like this:

```
                                        2
# B 4A20          OR V32
#
```

After the second # sign type: J 4A00. The screen will go blank. Now, type your digit, 5. A brief flash of the message is seen, then the screen is cleared and the # sign appears at the top.

The program has stopped at the location where we put our breakpoint (4A20), and now we can examine the registers.

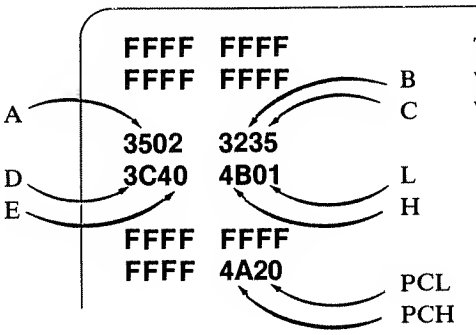
Type: R for a display of the registers in the format shown below:

```
A'   F'    B'   C'
D'   E'    H'   L'

A    F     B    C
D    E     H    L

IXH  IXL   IYH  IYL
SPH  SPL   PCH  PCL
```

This is what my register display looked like:



The FFFF values may vary.

The registers that concern us, at the present, are:

A the accumulator
B the register where our digit should be (ASCII)
C, D, E, H, L
PCH
PCL the program counter (high and low addresses)

The BREAK command stopped the program at memory location 4A20 as shown by the values in the program counter registers (PCH and PCL). Looking back at the program again, we must consider what has been done immediately prior to this spot in the program.

> The contents of the accumulator were moved into register B by the instruction at 4A1F *(supposedly!)*.

The register display shows that the value 35 (ASCII code for our digit) is in the accumulator, but the value 32 (ASCII code for 2) is in register B. *Something went wrong right there!* It may be a coincidence, but I noticc that register C contains the 35 we were looking for. *Stop* and *look up* the Z80 instruction for LD dst,src.

> For LD B,A the code should be 47. In our program it is 4F. *Somebody goofed!* That's error number one.

To correct this error:

1. Type: X to get out of the MEMORY mode.
2. Following the # sign, type: M 4A1F and sure enough, the display shows our error:

```
# M 4A1F 4F
```

3. Type the correction: 47 and the display shows:

```
# M 4A1F 4F 47
4A20 CD
```

Correction

Beginning of BREAK command still there. It must be removed next.

4. Type: X to exit the MEMORY Mode.
5. Type: F to remove the BREAKPOINT.

6. Type: J 4A00 to try the program again.

When we try the program again, the screen goes blank as it should. If we type in a 5 again, the display shows:

```
                                    5
THE ASCIIUCODE FOR V35
```

Something is still wrong! Since we're not getting the first part of our message correct, I suspect that we still have errors in LOOP2. Let's put in a break further down in the same loop. Let it pass through the loop one complete time.

Press the @ key to get back to T-BUG.

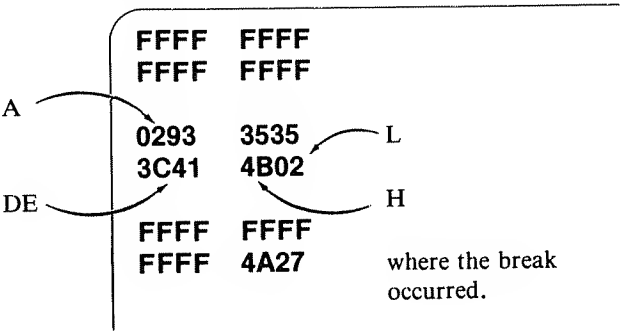
After the # sign type: B 4A27.

Then after the next # sign type: J 4A00. The screen goes blank.

Type in your digit, 5. The screen blanks with the # sign at the top.

Type in the REGISTER command:

The display shows:



Just before our breakpoint, the contents of register L were moved into the accumulator. Notice from the display that this was done correctly. Also the DE and HL register pairs have been incremented correctly. It seems that this loop is working as it is supposed to. Let's move on.

We should probably look next at what happens between the exit of LOOP2 and the beginning of LOOP3. This would call for a BREAK to be inserted at 4A2D, the beginning location for LOOP3.

If you are still in the REGISTER mode, type: X.

Then remove the present BREAK (at 4A27) by typing: F.

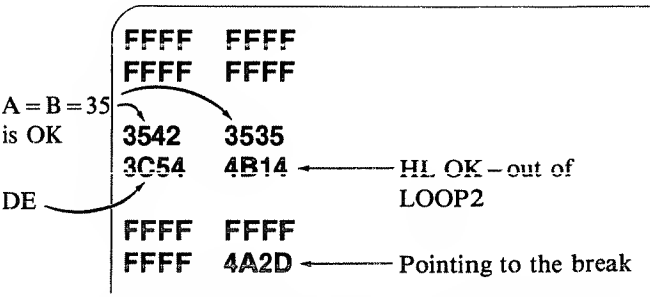
Now, type: B 4A2D.

Then, type: J 4A00.

When the screen goes blank, type in your digit, 5.

When the # sign comes up, type: R to look at the registers again.

Here's what my registers show:



The section that we are checking loads our digit (ASCII 35) into the accumulator from register B. It then puts it into the memory location pointed to by DE. We can see that 35 has been copied from register B into the accumulator. Since DE has been incremented and now reads 3C54, we should look at memory location 3C53 to see if the value, 35, was placed there as desired. Everything else seems to be correct.

Type: X to exit the REGISTER mode.

Then type: M 3C53

```
# M 3C53 35
```

Sure enough, it holds the value 35.

We look fine up to LOOP3.

Now type: X.

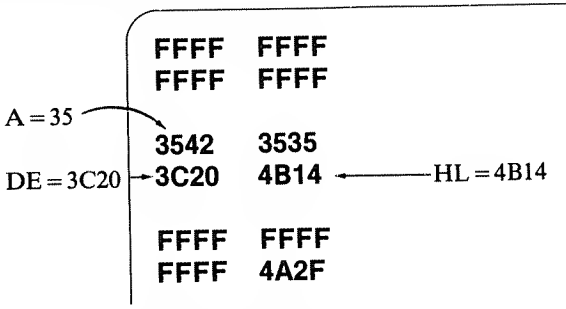
Then type: F.

Where do we look next? Let's check the first two instructions of LOOP3. That would be addresses 7A2D and 7A2E. Therefore the BREAK should be inserted at 4A2F. At this point our digit should have been printed. These two instructions should move the ASCII code for the letter "I" (from the word "IS") from the memory location pointed to by the HL register pair into the accumulator. Then the accumulator's contents should be placed in the display memory location pointed to by the DE register pair. To see if this has been done:

After the # sign, type: B 4A2F.

Then type: J 4A00.

When the screen goes blank, type your digit, 5.
When the # sign comes on again, type: R.
Here is what I see:



It's plain to see that the accumulator holds 35, the ASCII code for 5. *It should hold 20,* the ASCII value for a space, since 20 is in memory location 4B14. This space should have been displayed following our digit. HL is pointing to the right place; and if you examine memory location 4B14, you will see that 20 is there. From our previous BREAK, we know that 3C54 was in the DE register pair just before we came into this loop, and we have given no instruction to change it. *How did 3C20 get there? That is not correct.*

Looking at the instruction at location 4A2D, we see a

```
LD A,(HL)
```

Let's check the Z80 opcode for this instruction. It says 7E. *But,* we have a 5E in our program. The 5E instruction would move the contents of HL into register E, not the accumulator. Since 4B14 contains the value 20, this was moved into register E, not the accumulator. The instruction at 4A2E then moved the contents of the accumulator (still 35) into the memory pointed to by the register pair DE (3C20 instead of 3C54). That is why we saw that isolated number in the top row of the display.

Let's clear up that instruction now.
Type: X.
Then, type: M 4A2D.
The display shows the culprit:

```
# M 4A2D 5E
```

Change it by typing: 7E.

```
# M 4A2D 5E 7E
4A2E 12
```

Now type: X.
Then type: F.
Then: J 4A00.
When the screen goes blank, type your digit, 5.
Now we see:

```
THE ASCII CODE FOR 5 IS35
```

That looks fine—except that we seem to be missing a space between **IS** and **35**. The message completed in LOOP3 should read: "**THE ASCII CODE FOR 5 IS** ". There should be a space after the word "IS". The missing space may be discovered if we put a break at the exit of LOOP3. Let's try a BREAK at 4A37.

Type: @ to get to T-BUG.
After the #, type: B 4A37.
After the next #, type: J 4A00.
When the screen goes blank, type: 5.
When the # appears, type: R.
Now my display shows:

```
                FFFF  FFFF
                FFFF  FFFF
A=17 ──┐
       └─►      1742  3535
DE=3C57 ►3C57   4B17 ◄──── HL=4B17

                FFFF  FFFF
                FFFF  4A37
```

On the last pass through LOOP3, something is displayed (either the "S" in "IS" or a space after the "S"). Then DE and HL are incremented. Therefore HL should point to one memory location following the data displayed. But it can be seen that HL now points to the location where our last space is located. The accumulator, which now holds 17 (the current value of L), verifies this also. We don't really want to come out of the loop until HL points to 4A18.

Our compare value at memory location 4A33 is 17. The space missing is in location 4B17. Since the compare instruction and

the JUMP NZ instruction cause an exit to the loop when HL has been incremented to 4B17, we never get the last ASCII code read in. We should change the value in 4A33 to 18 (one more than the last location of the message).

After the # sign, type: M 4A33.

The display reads:

```
# M 4A33 17
```

Type: 18.

The display now shows:

```
# M 4A33 17 18
4A34 CD
```

Type: X to exit

After the #, type: F to remove the BREAK.

After the next #, type: J 4A00.

The screen goes blank. Type: 5.

And there, at last, is the correct message:

```
THE ASCII CODE FOR 5 IS 35
```

Now that we know that the program works for the digit 5:

Type: ENTER and input a new digit.

Repeat this for all ten digits to be sure that it works for all of them.

Type: @ when you are tired of the whole thing.

Now, you know how to debug all those errors which you are sure to put into your future programs. Don't be discouraged. Every programmer makes mistakes. The better you become at finding your errors quickly, the better programmer you will become. T-BUG can be a great aid for those who enjoy working in machine language.

SUMMING UP

We have merely scratched the surface of machine language programming in this book. However, you should now be familiar with the capabilities of T-BUG. It is a simple, but quite useful, tool which can be used to your advantage when creating your own programs in machine language.

Although machine language programming may seem much more demanding and much slower than programming in a high-level language, such as BASIC, programs can be much more efficient and executed in a much shorter time. If you don't believe this, try writing a program which will display the graphics of Program 5B using your Level I or Level II BASIC. Then compare the time it takes the TRS-80 to "draw" the three figures on the screen from your BASIC program and from your machine language program.

Your programming abilities will only improve with continual use. Experiment with the Z80 instruction set. We have only used a few instructions so that the programs would be easily understood. Work with short program segments, saving them on tape. Then, combine the short modules into larger, more complex programs.

If you have Level II BASIC, the Radio Shack Editor/Assembler package is a good investment for machine language programmers. It can be used to write, edit and assemble machine language programs from their mnemonic codes. The assembler keeps track of all the memory locations and other tedious details. The use of an assembler is the next step up for a serious machine language programmer.

APPENDIX
Tables

Table I Decimal-Hex Conversion

DEC.	HEX.	DEC.	HEX.	DEC.	HEX.
01	01	41	29	81	51
02	02	42	2A	82	52
03	03	43	2B	83	53
04	04	44	2C	84	54
05	05	45	2D	85	55
06	06	46	2E	86	56
07	07	47	2F	87	57
08	08	48	30	88	58
09	09	49	31	89	59
10	0A	50	32	90	5A
11	0B	51	33	91	5B
12	0C	52	34	92	5C
13	0D	53	35	93	5D
14	0E	54	36	94	5E
15	0F	55	37	95	5F
16	10	56	38	96	60
17	11	57	39	97	61
18	12	58	3A	98	62
19	13	59	3B	99	63
20	14	60	3C	100	64
21	15	61	3D	101	65
22	16	62	3E	102	66
23	17	63	3F	103	67
24	18	64	40	104	68

25	19	65	41	105	69
26	1A	66	42	106	6A
27	1B	67	43	107	6B
28	1C	68	44	108	6C
29	1D	69	45	109	6D
30	1E	70	46	110	6E
31	1F	71	47	111	6F
32	20	72	48	112	70
33	21	73	49	113	71
34	22	74	4A	114	72
35	23	75	4B	115	73
36	24	76	4C	116	74
37	25	77	4D	117	75
38	26	78	4E	118	76
39	27	79	4F	119	77
40	28	80	50	120	78

Table II Z80 Instructions and Opcodes

INSTRUCTION	OPERAND	HEX OPCODE
ADC	A,yy	CE yy
ADC	A,(HL)	8E
ADC	HL,rp	ED 01xx1010
ADC	A,(IX + d)	D0 8E yy
ADC	A,(IY + d)	FD 8E yy
ADC	A,r	10001xxx
ADD	A,yy	C6 yy
ADD	A,(HL)	86
ADD	HL,rp	00xx1001
ADD	A,(IX + d)	DD 86 yy
ADD	A,(IY + d)	FD 86 yy
ADD	IX,pp	DD 00xx1001
ADD	IY,rr	FD 00xx1001
ADD	A,r	10000xxx
AND	yy	E6 yy
AND	(HL)	A6
AND	(IX + d)	DD A6 yy
AND	(IY + d)	FD A6 yy
AND	r	10100xxx
BIT	b,(HL)	CB 01bbb110

Table II (Continued)

INSTRUCTION	OPERAND	HEX OPCODE
BIT	b,(IX + d)	DD CB 01bbb110
BIT	b,(IY + d)	FD CB 01bbb110
BIT	b,r	CB 01bbbxxx
CALL	llhh	CD 11 hh
CALL	NZ,llhh	C4 ll hh
CALL	Z,llhh	CC ll hh
CALL	NC,llhh	D4 ll hh
CALL	C,llhh	DC ll hh
CALL	PO,llhh	E4 ll hh
CALL	PE,llhh	EC ll hh
CALL	P,llhh	F4 ll hh
CALL	M,llhh	FC ll hh
CCF		3F
CP	yy	FE yy
CP	(HL)	BE
CP	(IX + d)	DD BE yy
CP	(IY + d)	FD BE yy
CP	r	10111xxx
CPD		ED A9
CPDR		ED B9
CPI		ED A1
CPIR		ED B1
CPL		2F
DAA		27
DEC	(HL)	35
DEC	IX	DD 2B
DEC	IY	FD 2B
DEC	(IX + d)	DD 35 yy
DEC	(IY + d)	FD 35 yy
DEC	rp	00xx1011
DEC	r	00xxx101
DI		F3
DJNZ	yy	10 yy
E1		FB

Table II (Continued)

INSTRUCTION	OPERAND	HEX OPCODE
EX	AF,AF'	08
EX	DE,HL	EB
EX	(SP),HL	E3
EX	(SP),IX	D0 E3
EX	(SP),IY	FD E3
EXX		D9
HALT		76
IM	0	ED 46
IM	1	ED 56
IM	2	ED 5E
IN	A,(yy)	DB yy
IN	r,(r)	ED 01xxx000
INC	(HL)	34
INC	IX	D0 23
INC	IY	FD 23
INC	(IX + d)	DD 34 yy
INC	(IY + d)	FD 34 yy
INC	rp	00xx0011
INC	r	00xxx100
IND		ED AA
INDR		ED BA
INI		ED A2
INIR		ED B2
JP	llhh	C3 ll hh
JP	(HL)	E9
JP	(IX)	DD E9
JP	(IY)	FD E9
JP	C,llhh	DA ll hh
JP	NC,llhh	D2 ll hh
JP	Z,llhh	CA ll hh
JP	NZ,llhh	C2 ll hh
JP	PO,llhh	E2 ll hh
JP	PE,llhh	EA ll hh
JP	P,llhh	F2 ll hh
JP	M,llhh	FA ll hh
JR	yy	18 yy

Table II (Continued)

INSTRUCTION	OPERAND	HEX OPCODE
JR	C,yy	38 yy
JR	NC,yy	30 yy
JR	Z,yy	28 yy
JR	NZ,yy	20 yy
LD	r,r	01dddsss
LD	r,yy	00xxx110 yy
LD	r,(HL)	01xxx110
LD	r,(IX + d)	DD 01xxx110
LD	r,(IY + d)	FD 01xxx110
LD	(HL),r	01110xxx
LD	(IX + d),r	DD 01110xxx yy
LD	(IY + d),r	FD 01110xxx yy
LD	(HL),yy	36 yy
LD	(IX + d),yy	DD 36 yy yy
LD	(IY + d),yy	FD 36 yy yy
LD	A,(BC)	0A
LD	A,(DE)	1A
LD	A,(llhh)	3A ll hh
LD	(BC),A	02
LD	(DE),A	12
LD	(llhh),A	32 ll hh
LD	A,I	ED 57
LD	A,R	ED 5F
LD	I,A	ED 47
LD	R,A	ED 4F
LD	rp,llhh	00xx0001 ll hh
LD	IX,llhh	DD 21 ll hh
LD	IY,llhh	FD 21 ll hh
LD	HL,(llhh)	2A ll hh
LD	rp,(llhh)	ED 01xx1011 ll hh
LD	IX,(llhh)	DD 2A ll hh
LD	IY,(llhh)	FD 2A ll hh
LD	(llhh), HL	22 ll hh
LD	(llhh),rp	ED 01xx0011 ll hh
LD	(llhh),IX	DD 22 ll hh
LD	(llhh),IY	FD 22 ll hh
LD	SP,HL	F9
LD	SP,IX	DD F9

Table II (Continued)

INSTRUCTION	OPERAND	HEX OPCODE
LD	SP,IY	FD F9
LDD		ED A8
LDDR		ED B8
LDI		ED A0
LDIR		ED B0
NEG		ED 44
NOP		00
OR	yy	F6 yy
OR	(HL)	B6
OR	(IX + d)	DD B6 yy
OR	(IY + d)	FD B6 yy
OR	r	10110xxx
OTDR		ED BB
OTIR		ED B3
OUT	(C),r	ED 01sss001
OUT	(yy),A	D3 yy
OUTD		ED AB
OUTI		ED A3
POP	IX	DD E1
POP	IY	FD E1
POP	rp	11xx0001
PUSH	IX	DD E5
PUSH	IY	FD E5
PUSH	rp	11xx0101
RES	b,(HL)	CB 10bbb110
RES	b,(IX + d)	DD CB yy 10bbb110
RES	b,(IY + d)	FD CB yy 10bbb110
RES	b,r	CB 10bbbxxx
RET		C9
RET	NZ	C0
RET	Z	C8
RET	NC	D0
RET	C	D8
RET	PO	E0
RET	PE	E8

Table II (Continued)

INSTRUCTION	OPERAND	HEX OPCODE
RET	P	F0
RET	M	F8
RETI		ED 4D
RETN		ED 45
RL	(HL)	CB 16
RL	(IX + d)	DD CB yy 16
RL	(IY + d)	FD CB yy 16
RL	r	CB 00010xxx
RLA		17
RLC	(HL)	CB 06
RLC	(IX + d)	DD CB yy 06
RLC	(IY + d)	FD CB yy 06
RLC	r	CB 00000xxx
RLCA		07
RLD		ED 6F
RR	(HL)	CB 1E
RR	(IX + d)	DD CB yy 1E
RR	(IY + d)	FD CB yy 1E
RR	r	CB 00011xxx
RRA		1F
RRC	(HL)	CB 0E
RRC	(IX + d)	DD CB yy 0E
RRC	(IY + d)	FD CB yy 0E
RRC	r	CB 00001xxx
RRCA		0F
RRD		ED 67
RST	p	11xxx111
SBC	A,yy	DE yy
SBC	A,(HL)	9E
SBC	HL,rp	ED 01xx0010
SBC	A,(IX + d)	DD 9E yy
SBC	A,(IY + d)	FD 9E yy
SBC	r	10011xxx
SCF		37
SET	b,(HL)	CB 11bbb110
SET	b,(IX + d)	DD CB yy 11bbb110

Table II (Continued)

INSTRUCTION	OPERAND	HEX OPCODE
SET	b,(IY + d)	FD CB yy 11bbb110
SET	b,r	CB 11bbbxxx
SLA	(HL)	CB 26
SLA	(IX + d)	DD CB yy 26
SLA	(IY + d)	FD CB yy 26
SLA	r	CB 00100xxx
SRA	(HL)	CB 2E
SRA	(IX + d)	DD CB yy 2E
SRA	(IY + d)	FD CB yy 2E
SRA	r	CB 00101xxx
SRL	(HL)	CB 3E
SRL	(IX + d)	DD CB yy 3E
SRL	(IY + d)	FD CB yy 3E
SRL	r	CB 00111xxx
SUB	A,(HL)	96
SUB	A,(IX + 2)	DD 96 yy
SUB	A,(IY + 2)	FD 96 yy
SUB	A,r	10010xxx
SUB	A,yy	D6 yy
XOR	yy	EE yy
XOR	(HL)	AE
XOR	(IX + d)	DD AE yy
XOR	(IY + d)	FD AE yy
XOR	r	10101xxx

Symbols used in Table II:

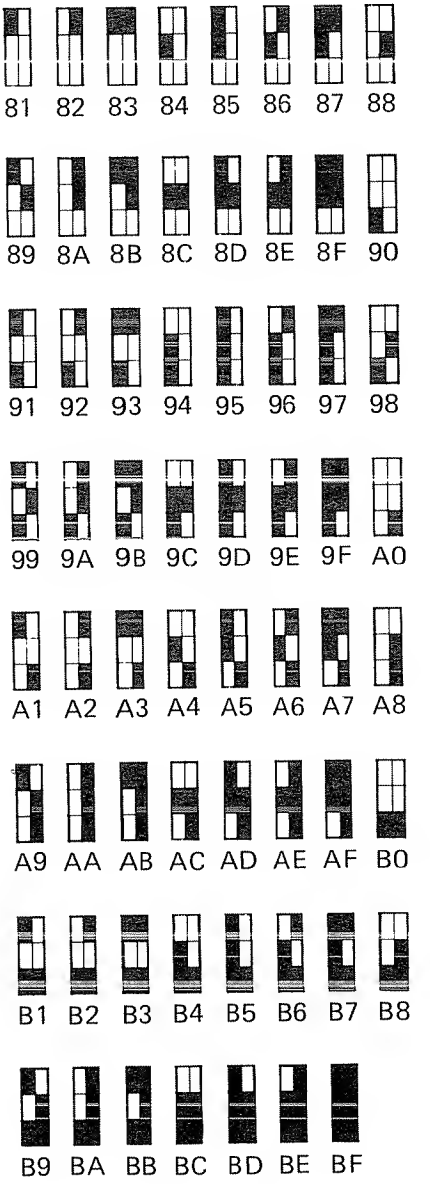
d = displacement
p = restart code
r = register code
x = binary bit
y = hex digit

bbb = bit code
ddd = destination register code
sss = source register code
hh = high-order hex digits
ll = low-order hex digits
pp = (contents of) register pair code pp
rp = register pair code
rr = (contents of) register pair code rr

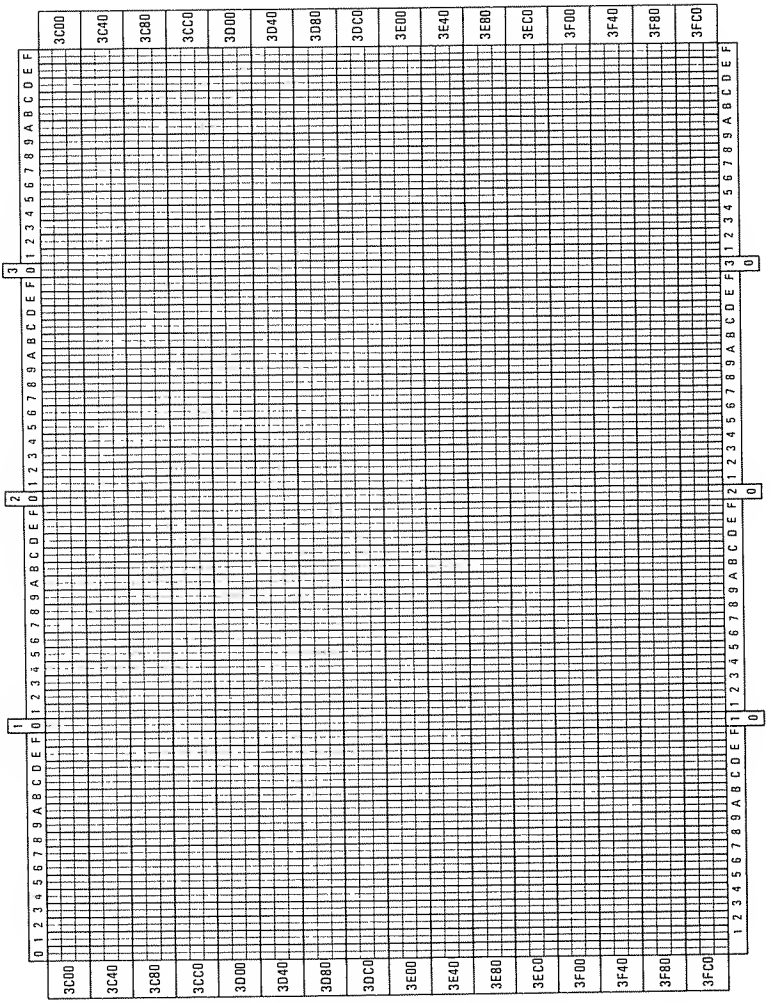
Table III Hex-ASCII

Hex LSD \ Hex MSD	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	=	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	-	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENO	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	=	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	•		J	Z	j	z
B	VT	ESC	+	:	K	[	k	{
C	FF	FS	.	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS		>	N	Λ	n	~
F	SI	US	/	?	O	—	o	DEL

Table IV Graphic Characters and Codes



TRS-80 Video Display Worksheet



INDEX

About the Book . . .

This is the only book to describe in detail the machine language monitor operations of the popular Radio Shack TRS-80 computer. Each command is explained and discussed in detail and examples are given to show how the commands may be used. Each step of every sample program is accompanied by a sketch of the corresponding video display for complete "no question about it" understanding of the operations. The examples constitute practical applications which make this book not only instructional, but useful as well.

About the Authors . . .

Don Inman, who graduated from the University of Northern Iowa with a Bachelor's Degree in Mathematics is a long-time computer aficionado. When he's not busy teaching at a local High School, he's usually tied up in a computer text or some how-to-book project. Don is also the author of INTRODUCTION TO TRS-80 GRAPHICS and is presently working on yet another TRS-80 book for dilithium.

Kurt Inman is Don's 15 year old son and an author in his own right. He will collaborate with his dad on his next TRS-80 book as well.

Other dP books on the TRS-80

INTRODUCTION TO TRS-80 GRAPHICS
by Don Inman

This book is the fastest-selling dP title ever and readers from every sector have praised its broad scope and ease of understanding. It begins with the basic concept on line drawing then moves into geometric shapes, moving figure animation and more advanced projects. An understanding of BASIC is required.

ISBN 0-918398-18-5 **$8.95**

MICROSOFT BASIC
by Ken Knecht

Another quick selling dP title, this book is particularly useful for TRS-80 owners, but can be used with any machine using the MITS family of BASIC interpreters. This book supplies a useful intro and tutorial in Microsoft BASIC and illustrates concepts with examples that will actually run on your computer.

ISBN 0-918398-23-1 **$8.95**

ISBN 0-918398-33-9

dilithium Press
30 NW 23rd Place
Portland, Oregon 97210